Office de la Propriété Intellectuelle du Canada
Un organisme d'Industrie Canada

Canadian Intellectual Property Office
An agency of Industry Canada

CA 2436763 A1 2005/02/08

(21) 2 436 763

(12) DEMANDE DE BREVET CANADIEN
CANADIAN PATENT APPLICATION

(13) A1

(22) Date de dépôt/Filing Date: 2003/08/08

(41) Mise à la disp. pub./Open to Public Insp.: 2005/02/08

(51) Cl.Int.^{7}/Int.Cl.^{7} G06F 17/60, G06F 9/44, H04L 12/16

(71) Demandeur/Applicant:
SILVER LEAP TECHNOLOGIES INC., CA

(72) Inventeurs/Inventors:
CREPIN, VINCENT, CA;
DELAGE, CLAUDE, CA;
HAMEL, ANDRE, CA;
ROY, PASCAL, CA

(74) Agent: BROUILLETTE KOSIE PRINCE

(54) Titre : SYSTEME ET METHODE PERMETTANT DE REALISER DES APPLICATIONS WEB
(54) Title: SYSTEM AND METHOD TO DEVELOP WEB BASED APPLICATIONS

Silver Leap
Technologies

The ACES Integrated Environment

An agile Application Lifecycle Management Solution integrating BPM, SOA, and MDA.

Technical White Paper

CONTENTS

1 ABSTRACT

1.1 Executive Summary

This paper presents Silver Leap's vision of the ACES Integrated Environment, an agile J2EE compliant Application Lifecycle Management (ALM) solution for enterprise application development.

The ACES Integrated Environment builds on industry trends such as ALM, Business Process Management (BPM), Model Driven Architecture (MDA), SOA (Service Oriented Architecture) and Agile development processes. The solution integrates existing SilverLeap, SunONE and open source technologies with the addition of new innovative easy to use features that are directly addressing current software development community issues and problems.

With its expertise and current technologies in ARAD (Architected Rapid Application Development) tools, SivlerLeap is in a unique position to allow Sun to deliver quickly and effectively a next generation software development environment platform capable of retaining Sun's current software developer base and augmenting it at the expense of its competitors.

1.2 Key Issues

What major problems is the AD community currently facing?

What are the major initiatives that are currently under way in the AD community to solve those problems?

What are BPM, MDA, SOA and agile processes and how can they be tightly integrated into a simple yet complete ALM solution?

What would the solution look like in the context of Sun's current SunONE and Orion strategies?

Why is Silver Leap the best fit for Sun to deliver this solution in a timely and cost effective manner?

1.3 Audience

This white paper is targeted primarily at Sun's Software Development Executives. The elements of the solution presented in this paper address important strategic trends in the software development tool market. Additionally, section 7 delves a little deeper in technical details so Sun engineers can do a preliminary technical assessment of the proposed solution.

2 INTRODUCTION

2.1 Current State of Application Development

As application software development complexity continues to grow exponentially and with the increasing pressure on IT to deliver more business value quickly and efficiently, it is becoming clear that the traditional software development methodologies, processes, techniques and tools can no longer keep up.

The now famous 1994 Standish Group Chaos study of Large Government and Commercial systems indicated that

- Only 16% of the projects were completed on time & under budget
- 31% were canceled
- 53% of the projects exceeded their budget by more than 100%
- Of the 16% completed project, only 61% of specified features were included

Since then, other more recent, but smaller studies on IT software development projects have shown evidence that the trend is really not improving with time.

Some of the key problems behind those dismal numbers include:

- Increasing complexity of applications and platforms
- Mismatch between what is needed by business and what is actually delivered by IT
- Fast changing business needs
- Ensuring architectural integrity across large projects
- Rapid pace of introduction of new technologies
- Finding and retaining qualified labor
- Integrating legacy applications and databases
- Reusing existing components in a different configuration
- Multiple competing middleware systems
- Adapting shrink-wrap products to the business (ERP, CRM…)
- Lack of real feedback during development

2.2 AD Trends

A number of trends have surfaced fairly recently to address some of those pressing problems:

ALM: Application Lifecycle Management is a one-stop-shop for application development. An ALM solution integrates the necessary tools to address the entire application lifecycle such as definition, design, development, testing, deployment and management. It accelerates product delivery and increases ROI by improving team productivity.

BPM: Business Process Management allows the business to easily define (or model), modify and monitor the business processes so changing business needs get addressed very quickly while diminishing the overhead of translating business needs into an actual IT implementation. Currently, BPM and system development are fairly disjoint processes. Current BPM solutions assume software systems already exist and rely on Enterprise Application Integration (EAI/B2Bi) technologies to connect business processes to actual systems.

MDA: Model Driven Architecture is an attempt to raise the level of abstraction of system development in order to deal with increasing complexity. By working at the higher levels of abstraction of models, developers can concentrate on implementing business value, leaving the complexity of mapping the models to platform specific implementations to automated transformation tools. These tools encapsulate knowledge of common architectural patterns, best practices and technology mappings. MDA is strictly concerned with the modeling of software systems. The responsibility of modeling business processes and rules is the domain of BPM.

SOA: Service Oriented Architecture is a type of enterprise architecture. It embodies the old concept of reuse but at a higher level. Component reuse at lower levels has largely been unsuccessful in the industry. SOA attempts to raise the reusability at the level of business processes or domain level. Web Services is an embodiment of SOA using web technologies. It allows organizations to package business functionality and make it available either internally or externally for consumption through a set of well-defined and standard web interfaces.

Agile Processes: Development methodologies like XP, Scrum and others are becoming increasingly popular, allowing business and IT to work in a tighter feedback loop to ensure business needs are being addressed properly and

quickly. In some sense, agile processes try to merge the activities of defining the business processes and defining the supporting software systems to achieve better synergy and efficiency.

2.3 Silver Leap's Solution

The *ACES Integrated Environment* is Silver Leap's solution to address the current problems of the industry. It is an agile ALM suite of tools covering the entire application software development lifecycle. It raises the level of abstraction for developing applications by embracing the OMG's Model Driven Architecture and seamlessly integrating Business Process Modeling concepts. Its RAD technology allows the easy and rapid creation of applications and services, fully supporting iterative and agile development processes.

2.4 Overview of document

Section 3 describes briefly ALM and lists the typical ALM activities and tools.

Section 4 discusses Business Modeling and it's role and impact on the software development process.

Section 5 presents SOA and it's importance as a platform for reuse.

Section 6 presents the OMG's MDA initiative.

Section 7 discusses the details of Silver Leap's proposed ALM solution.

Section 8 is presents the conclusion of this paper.

Section 9 is a glossary of terms used in the paper.

Appendix A presents more details about Business Modeling.

Appendix B presents more details about MDA.

3 APPLICATION LIFECYCLE MANAGEMENT

3.1 Definition

Application Lifecycle Management (ALM) embodies all the activities required to bring a software application or software system to life.

3.2 Activities

ALM activities usually boil down to the following:

Definition: state the problem and the functionality of the system that will solve the user's problems

Design: state how the system will implement the required functionality

Development: implement the system

Testing: verify that the system actually solves the user's problems

Deployment: make the system available to its users

Management: coordinate all the activities

3.3 Tools

Here is a list of typical tools that make up an ALM suite:

Activity	Tools
Definition	• Software Requirements Manager
Design	• Software System Modeler
Development	• Integrated Development Environment (IDE) • Libraries/Frameworks
Testing	• Testing tool (Unit/Integration/System) • Profiler • Code analysis & metrics
Deployment	• Web/Portal/Directory/Identity Management Servers • Application/Integration/Message Queuing Servers • Databases
Management	• Project Management • Software Configuration Management ▪ Software Asset Management ▪ Defect & Change Tracking

3.4 Scope

As of today, most ALM solutions deal only with the software system lifecycle. An upstream activity such as Business Process Modeling that provides input into the definition of the software system is typically not included. However, looking at the MDA roadmap and at the upcoming technologies included in UML 2.0, it is clear that BPM will become an integral part of an ALM solution in the future.

4 BUSINESS MODELING

4.1 Definition

With ever-increasing competition for markets, jobs and scarce resources, businesses today can only survive by constantly improving their quality, efficiency and cost-effectiveness. One way of assuring this constant level of excellence is by modeling the business.

Business modeling (BM), an activity of Business Process Management (BPM) consists in documenting a business. Although there are numerous ways of doing this, the motivations behind it are always the same:

- To understand the key mechanisms of an existing business.
- To try to improve existing business processes or create new ones.
- To act as the basis for creating suitable information systems supporting the business.
- To facilitate the integration of the business with others or to facilitate the integration of sub-parts of the business.
- To try to automate parts of the business.
- To have the possibility to monitor the business by providing metrics.

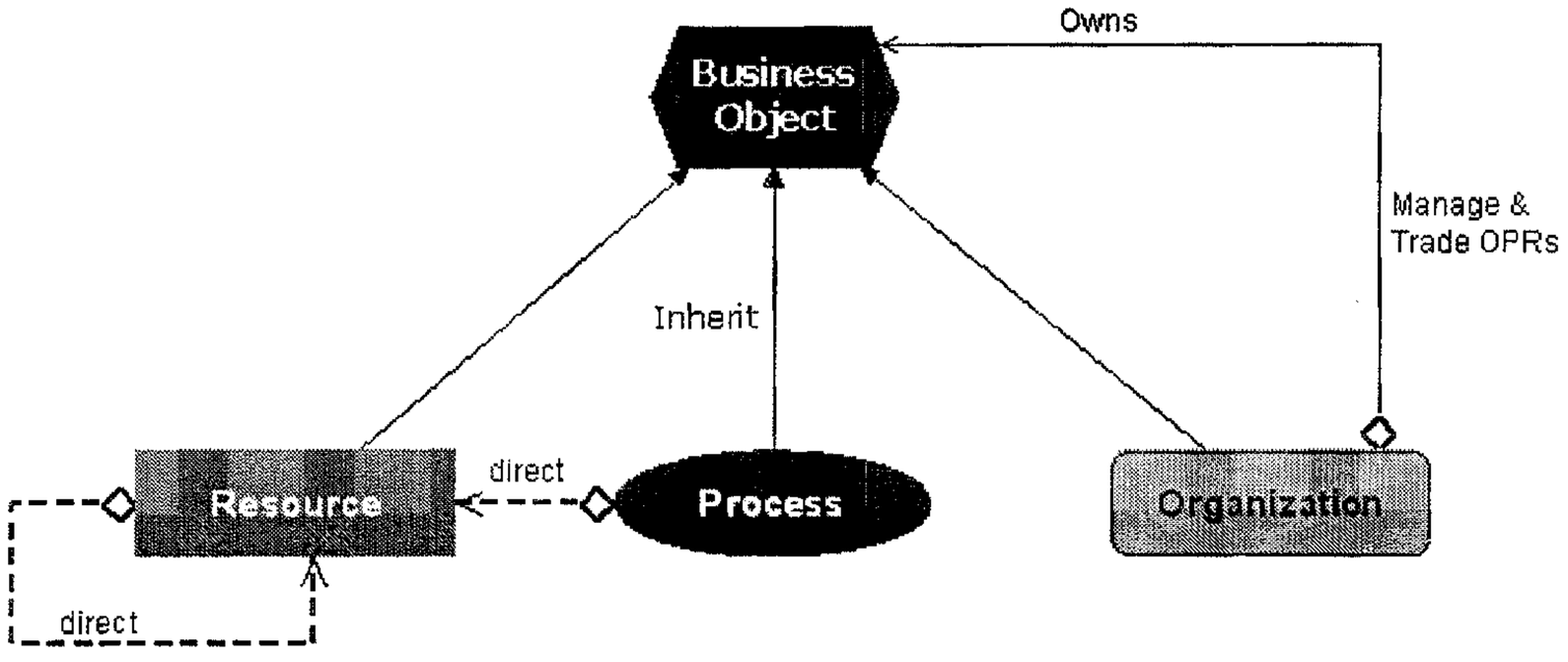
An enormous market has emerged around BM supporting methodologies and tools, mostly in Europe. Numerous methods have tried to abstract the concepts of business modeling. For example, Eriksson and Penker [ERIK 2000] use the following concepts to describe the business system:

- Resources: the objects within the business
- Processes: the activities performed within the business
- Goals: the purpose of the business
- Rules: statements that define or constrain some aspect of the business

We can find that all existing methodologies share similar concepts.

Based on years of empirical research, IT architects have been able to identify a reduced set of design elements, or design abstractions, to significantly simplify many aspects of design and the design process. This discovery is analogous to the discovery of the Reduced Instruction Set Computing (RISC) in computer hardware design, which was recognized as a desirable alternative to Complex Instruction Set Computing (CISC) in most situations. Reduced Abstraction Set Computing (RASC) constitutes a high-level language that can be shared by both

business and IT designers. These abstractions are Organization, Process and Resource. Many methodologies use these abstractions successfully today as Convergent Engineering and the respected Advanced Network System Architecture (ANSA) (Iggulden, 1994) the predecessor and principal basis for ISO Open Distributed Processing (ODP) standards. The following figure illustrates these abstractions:



4.1.1 Organization

Organizations manage processes, resources and other organizations. They group people and other resources charged with carrying out specific business processes.

4.1.2 Resources

Resources are intelligent units of value, cost and action in an organization. They represent sources of business value, work and information used by other abstractions.

4.1.3 Business Processes

A business process is a collection of activities that takes one or more kinds of input and creates an output for a particular customer or market. It implies a strong emphasis on how work is done within an organization, in contrast to a product's focus on what. A process is thus a specific ordering of work activities across time and place, with a beginning, and end, and clearly identified inputs and outputs: a structure for action [DAVEN 1992].

It is a goal directed sequence of activities or tasks that are enabled by resources.

4.1.4 Business rules

A business model contains business rules that define constraints, conditions and policies for how the business processes are to be performed. Business rules affect all the other concepts; they can constrain the execution of a business process, the behavior of a resource, and the means to achieve a specific goal.

Business rules can be defined as:

- Declarations of policies or conditions that must be satisfied [OMG Analysis and Design Reference Model 1992]
- Units of business knowledge [Odel 1998]
- Statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behavior of the business [GUIDE Business Rules Project 1995].

In order to be executable, a UML model needs to describe the static and dynamic nature of the problem. The static nature is described using class diagram and OCL. The dynamic nature is described using state chart diagrams and ASL. For a primer on OCL and Action Semantics, see appendix A.1 and A.2.

As we will see later, there are ways to automate the creation of meaningful Business Models using the previous abstractions.

4.1.5 Using UML

Since its introduction in 1997, UML has quickly become the standard modeling language for software development, especially for projects that use object orientation. In its next release (2.0), UML will provide a fairly complete set of features to support business modeling. For a detailed description of the UML 2.0 features, we refer you to OMG's web site.

Having an object-oriented business model, which can be mapped easily to object technology is a valuable simplification. Using object-oriented techniques, the modeling concepts and the structure used in the business model can be the same as those used in the analysis models for the information systems.

Furthermore, UML is a standard notation driven by the OMG and we consider it to be fairly easy to learn and very well supported by a large amount of tools.

The main disadvantage of UML for this purpose is that it is not formal enough in the BM field to allow the automation of the different required steps. For example, business use cases and interaction diagrams (sequence) cannot be interpreted directly by a processor (ex: a code generator).

4.1.6 The different views of a Business

Most methodologies share a "Multiple View" approach into modeling a business. For example, the ARIS methodology uses the ARIS house analogy [DAVIS 2001], which consists of a 4-views-model: Organizational, Data, Control and Function. Other methodologies will use variants of these but we could summarize the views as follows:

- The Vision view that contains the objectives of the business
- The Organizational view that lists the different resources of the business
- The Process view that illustrates the activities and processes in the business.
- The Behavioral view that shows the behavior of the resources and processes of the business.

For a detailed discussion of these views, see appendix A.4.

4.2 Business Patterns

Many problems that recur when modeling businesses have been resolved before. Business patterns, in the same way that Design patterns do but at a different level, make it possible to capture and describe these business-modeling problems and their corresponding solutions so they can be reused. The architect Christopher Alexander defines a pattern in his book, *The Timeless Way of Building* [1979] as follows:

"Each pattern is a three-part rule, which expresses a relation between a certain context, a problem and a solution.

- As an element in the world, each pattern is a relationship between a certain context, a certain system of forces, which occurs repeatedly in that context, and a certain spatial configuration, which allows these to resolve themselves.
- As an element of language, a pattern is an instruction, which shows how this spatial configuration can be used, over and over again, to resolve the given system of forces, wherever the context makes it relevant.
- The pattern is, in short, at the same time a thing, which happens in the world, and the rule which tells us how to create that thing, and when we must create it. It is both a process and a thing; both a description of a thing that is alive, and a description of the process which will generate that thing."

To successfully apply a pattern, the area to which we want to apply it must be fully understood. Business patterns are used to create understandable and flexible business models that describe the structure and behavior of a business. Gamma [GAMMA, 1995] identifies 3 types of patterns as follows:

- Functional: providing solutions to functional problems
- Structural: providing solutions to structural issues such as how to structure resources
- Behavioral: applied to dynamic descriptions such as how something changes over time.

A lot of work has been done in the field of patterns. For example, Eriksson and Penker [ERIK 2000] present many business patterns in their book on Business Modeling in 3 categories: Resource and Rule patterns, Goal patterns and Process patterns. These patterns should be reused as much as possible in every business modeling activity and a tool that would allow the automation of this step would be highly desirable.

4.3 Workflow

In many cases, business modeling consists in defining workflows of documents in the business. This is a specialized case of business processes for which many tools exist on the market. It should be emphasized that creating a workflow is a subset of modeling a business process. A complete business modeling solution allows the creation of such workflows.

4.4 B2Bi (collaborating business processes)

A current trend for integrating businesses is to expose processes at the service level and use standard protocols to exchange data. Web services and Bizztalk are very well known initiatives to address these problems.

4.5 Business Process Management

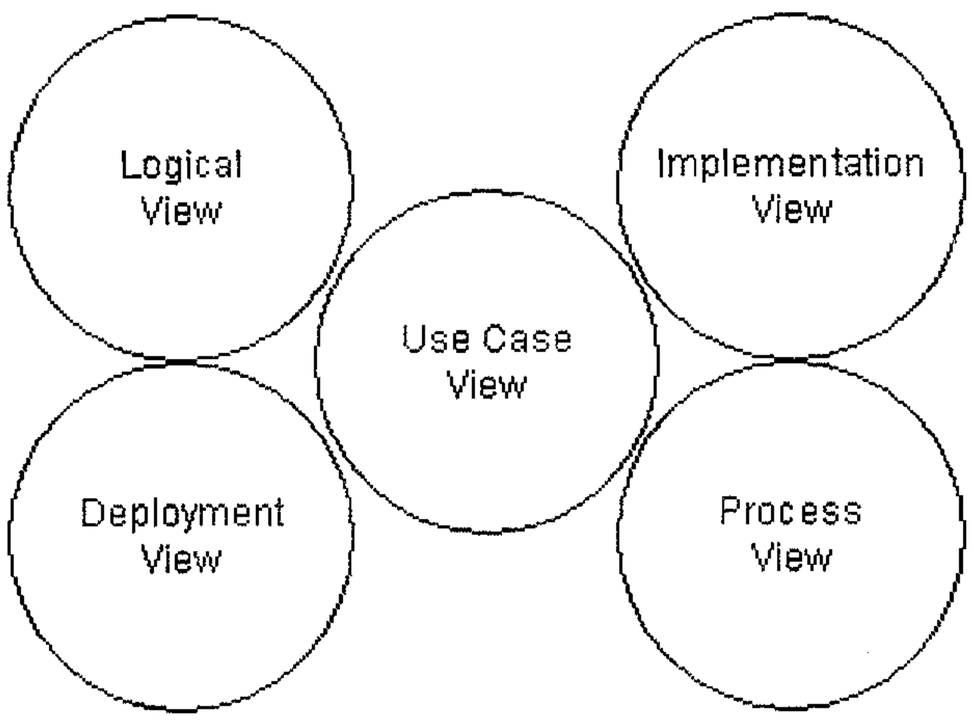
The Business Process Management encompasses all Business Process activities. Some of the most important activities include:

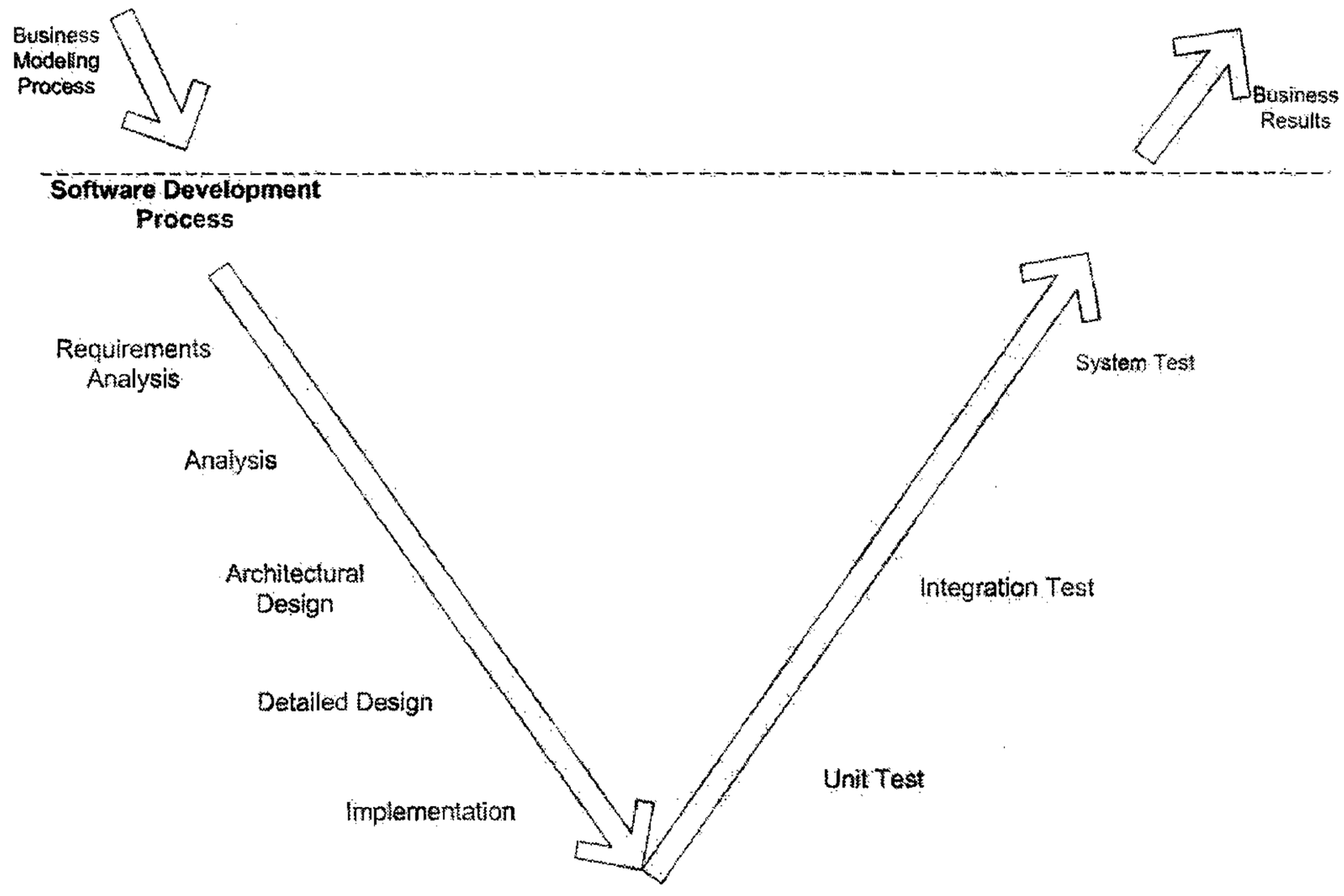
- Modeling
- Integration
- Orchestration
- Monitoring

4.6 From a Business model to a Software model

The most difficult part when starting from a business model to deduce the functional requirements of a software system is to obtain precise information from a "not too formal" language. This task is normally done by highly skilled analysts that must produce what is called a Software Architecture.

Building a software architecture is not a trivial task. Krutchen describes a 4+1 view of a software architecture that is becoming a de facto standard in the industry. His recommendation, the *4+1 View Model of Architecture,* is illustrated in the following figure:

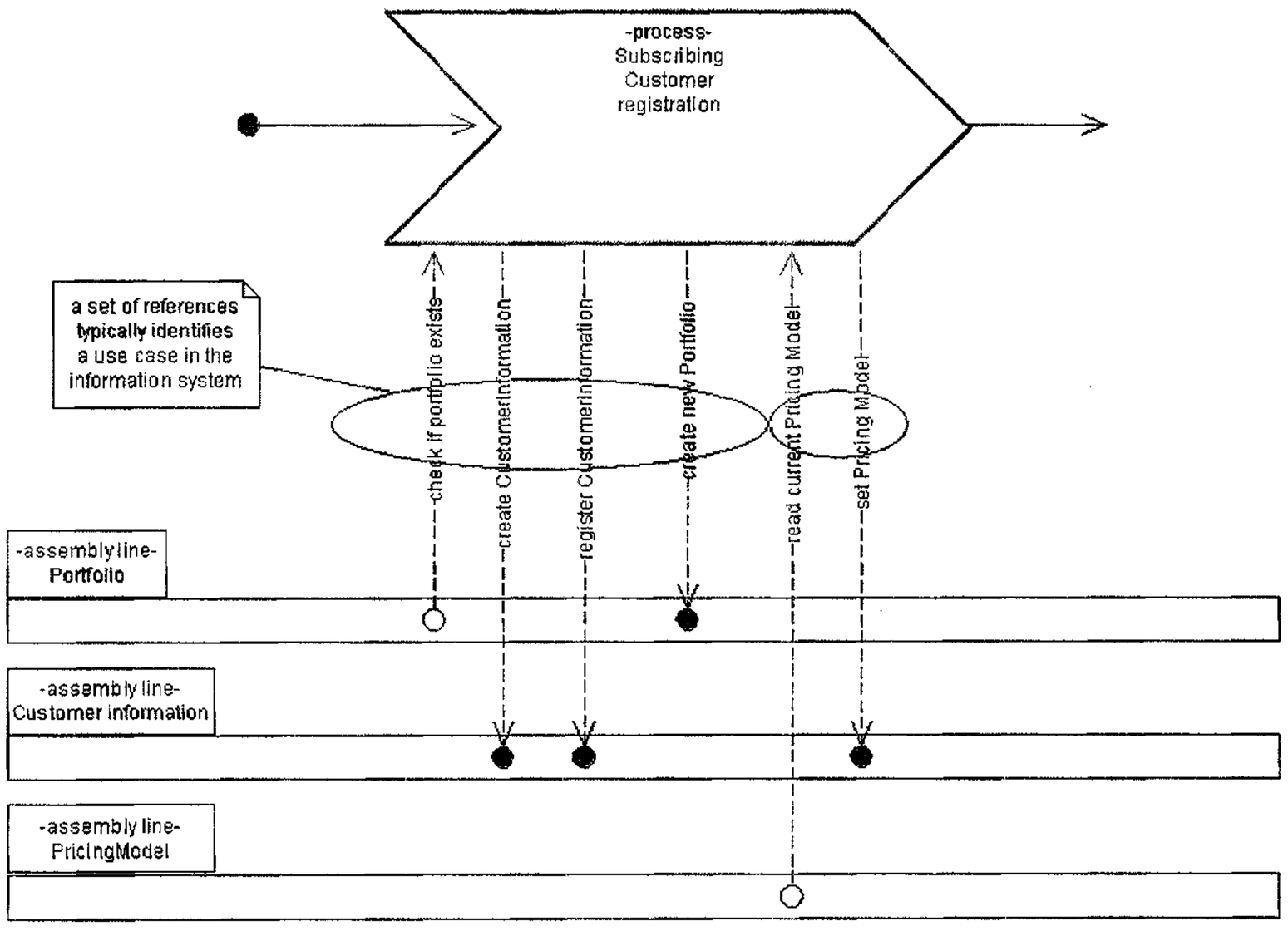




Traditionally, Business Modeling was done separately from other system development steps. As shown in the previous diagram, it was very often a prerequisite that a very stable Business Model existed before any system development could get under way. Modern iterative AD processes tend to integrate business modeling into the feedback loop in order to minimize risks and allow the developers to get the business user's feedback quickly and often.

So how is the information found in the business models transferred to the software models? The following actions summarize (this is by no way a general recipe) the roadmap to accomplish this [ERIK 2000]:

- Identify the information systems that best support the operation of the business. The process and the assembly line diagrams in the business model are used to identify activities that require services from an information system. Typically, these activities are: information storage/retrieval, processing, conversion, presentation, knowledge and decision-making, communication and control.
- Find functional requirements. Use cases can be considered as the specification of the services the software system provides. The assembly line diagram is used to identify the services a process needs from different systems. The following figure illustrates this:



- Find non-functional requirements. These are deduced from textual descriptions in the business model.
- Act as a basis (the business model) for analysis and design of the system. Much information can be deduced from the different process views as for example, the classes and the collaboration between different objects and the business rules.

All these tasks are not very straightforward. They require a great deal of work, intuition and experience and are usually done by highly skilled analysts. However, a lot of work has been done on the subject, numerous methodologies exist and are well documented.

To automate these steps, a kind of formalism has to be defined in the business model to facilitate the transformation to the software architecture. Using UML as the language to define the business model and the software architecture is a first step in the right direction and also what MDA prescribes. But we should go further. An ontology [JENZ 2003] can be used to establish the bridge between the world of natural language and the world of machines. Essentially, the business model is expressed in a language neutral knowledge base customized to the needs of the business. The ontology provides a way to establish a common language so information can be exchanged between the business model and the software architecture.

Because both types of models speak the same language, it is much easier for automated tools to automate the transformations between these 2 models. In conclusion, "Ontology-driven software artifact generation and the OMG's Model-Driven Architecture (MDA) are complementary approaches that fit together perfectly well". [JENZ 2003]

The Convergent Engineering approach [TAYLO, 1995] is another way of bridging the business models and the software models. It is the cornerstone of simplification and the key to resolving many of the pathologic problems suffered by IT organizations. It drastically simplifies both business and software development by consolidating the aspects they have in common. This approach prescribes the use of object-orientation to create models of the business and the software systems that support it. However, in that special case, some compromises have been done on the business model sides restricting the level of complexity attainable in these models. This restriction is essentially due to the tight mapping with information structures.

As we have seen previously, the use of business patterns can help "formalize" the business models but this use requires extensive knowledge and experience.

For a discussion on BM standards, see appendix A.3.

5 SERVICE-ORIENTED ARCHITECTURE (SOA)

5.1 Definition

With the introduction of Web Services over the last year or so, there has been a renewed interest in service-oriented architecture (SOA) [STEVENS 2003]. SOA is an architecture that has special properties. It is an architecture made up of components and interconnections that stress interoperability and location transparency. The term service has been used for more than two decades. For example, several leading transaction-monitoring software companies have used the term "service" in the early 1990s. Many client-server development efforts in the 90s used the term "service" to indicate the ability to make a remote method call. Web Services has given the term service more prominence in the last few months. Services and service-oriented architectures are really about designing and building systems using heterogeneous network addressable software components.

A service-oriented architecture is one that has a robust service layer. Most companies have already built components with many of these properties. They are some of the things to keep in mind when designing services or harvesting existing components to be service-enabled. Services can be built using a large number of technologies from application servers like .NET or J2EE, middleware solutions, or even adapters for database systems that enable access to data via Web services.

A service is behavior that is provided by a component for use by any other component based only on the interface contract. It has a network-addressable interface and may be dynamically discovered and used.

Web Services provide an implementation of SOA. They include HTTP as the primary network protocol, SOAP/XML for the payload format, UDDI for service registry, and WSDL to describe the service interfaces.

However, SOA does not require Web Services. SOA is a design and a way of thinking about building software components.

5.2 The Benefits of a Service-Oriented Architecture

In SOA, developers create services in a service layer. These services have published interfaces that support a distinct business domain. Organizations that focus their development effort around the creation of services will realize many benefits. The most common scenario for development organizations is to have some experience with component-based development. The use of application

servers such as J2EE or .NET for hosting applications is becoming more common. If your organization is using component-based development practices and application servers for business logic, then you are already service-oriented. By following the service-oriented mind set with even more rigor, combined with the component-based approach to software development, your organization will likely realize many benefits:

5.2.1 Better Return on Investment

The creation of a robust service layer has the benefit of a better return on the investment made in the creation of software. Services map to distinct business domains. For example, a company might create an inventory service that has all of the methods necessary to manage the inventory for the company. By putting the logic into a separate layer, the layer can exist well beyond the lifetime of any system it is composed into. For example, if your organization needs to create a credit card authorization service, there are basically two options. Developers will either develop the functionality as part of the application that needs it, or they will develop it as a separate component. If credit card authorization is developed as a separate component and used as a service, then it is likely to outlive the original application.

5.2.2 Code Mobility

Since location transparency is a property of a service-oriented architecture, code mobility becomes a reality. The lookup and dynamic binding to a service means that the client does not care where the service is located. Therefore, an organization has the flexibility to move services to different machines, or to move a service to an external provider.

5.2.3 More Security

The creation of a service layer by definition means that developers have created an additional network interface that can be used by multiple applications. When systems were built using monolithic or client-server methods, security was normally handled on the front-end. Companies often did not even implement database security because it is too hard to maintain multiple security lists. Services on the other hand are used by multiple applications, so they have their own security mechanisms. An application will therefore have multi-level authentication at both the client level and at the service level.

5.2.4 Better Testing/Fewer Defects

Services have published interfaces [1] that can be tested easily by developers by writing unit tests. Developers can use tools such as JUnit for creating test suites. These test suites can be run to validate the service independently from any application that uses the service. It is also a good practice to run the unit tests during an automated build process. There is no reason for a QA tester to test an application if the unit tests did not complete successfully. More and better testing usually means fewer defects and a higher overall level of quality.

5.2.5 Support for Multiple Client Types

As a benefit of a service-oriented architecture, companies may use multiple clients and multiple client types to access a service. A PDA using J2ME may access a service via HTTP, and a SWING client may access the same service via RMI. Since the layers are split into client and service layers, different client types are easier to implement.

5.2.6 Service Assembly

The services that developers create will evolve into a catalog of reusable services. Customers will come to understand this catalog as a set of reusable assets that can be combined in ways not conceived by their originators. Everyone will benefit from new applications being developed more quickly as a result of this catalog of reusable services.

5.2.7 Better Maintainability

Software archeology [2] is the task of locating and correcting defects in code. By focusing on the service layer as the location for your business logic, maintainability increases because developers can more easily locate and correct defects.

5.2.8 More Reuse

Code reuse has been the most talked about form of reuse over the last four decades of software development. Unfortunately, it is hard to achieve due to language and platform incompatibilities. Component or service reuse is much easier to achieve. Run-time service reuse is as easy as finding a service in the directory, and binding to it. The developer does not have to worry about compiler versions, platforms, and other incompatibilities that make code reuse difficult.

5.2.9 Better Parallelism in Development

The benefit of multiple layers means that multiple developers can work on those layers independently. Developers should create interface contracts at the start of a project and be able to create their parts independently of one another.

5.2.10 Better Scalability

One of the requirements of a service-oriented architecture is location transparency. To achieve location transparency, applications lookup services in a directory and bind to them dynamically at run-time. This feature promotes scalability since a load-balancer may forward requests to multiple service instances without the knowledge of the service client.

5.2.11 Higher Availability

Also because of location transparency, multiple servers may have multiple instances of a service running on them. If a network segment or a machine goes down, a dispatcher can redirect requests to another service without the client's knowledge.

5.3 Conclusion

A service layer takes little extra effort in the beginning of a project, but the benefits pay off in the long run. From higher quality to more reuse to better scalability and availability, the benefits of implementing a service layer far outweigh the cost and extra effort involved.

6 MDA – MODEL DRIVEN ARCHITECTURE

6.1 Definition

MDA or Model Driven Architecture is a new way of writing specifications and developing applications. It focuses first on the functionality and behavior of a distributed application or system, undistorted by idiosyncrasies of the technology or technologies in which it will be implemented. MDA divorces implementation details from business functions.

In the context of increasing complexity and the continuing proliferation of middleware "silver bullets" platforms and supporting technologies, MDA proposes an added level of abstraction providing software stability, investment protection and ROI. Platform independent models constitute a stake that remains fixed in the ground while the infrastructure around it shifts over time.

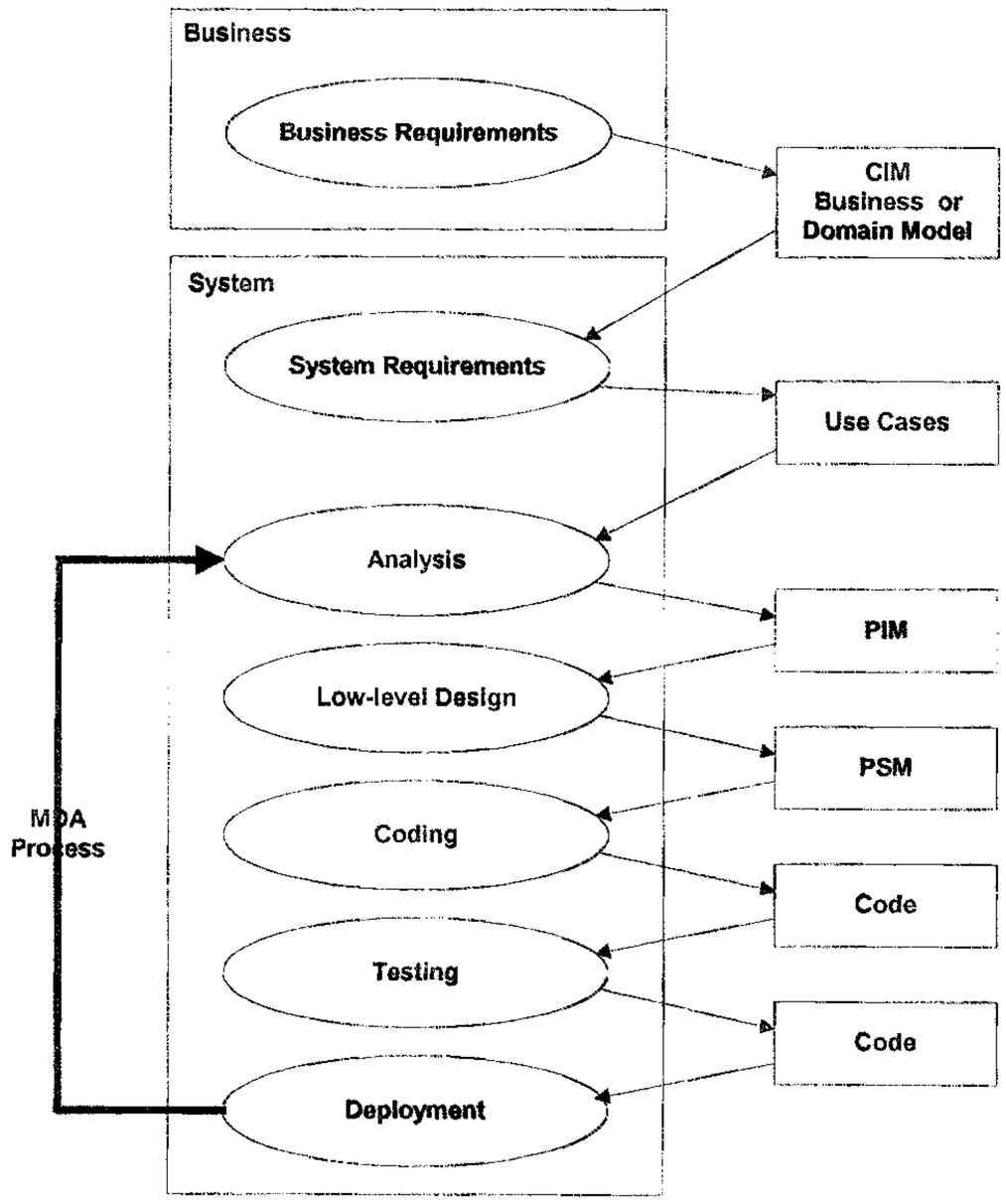
The MDA unites OMG's well established modeling standards (see Appendix B.1) with middleware technology - past, present, and future. Rather than focusing on yet another "next best thing," MDA raises the bar and designs portability and interoperability into the application at the model level.

6.2 Goals

MDA is designed to address those key problems. The three primary goals of MDA are **portability**, **interoperability** and **reusability** through architectural separation of concerns.

6.3 Life Cycle

6.3.1 Overview



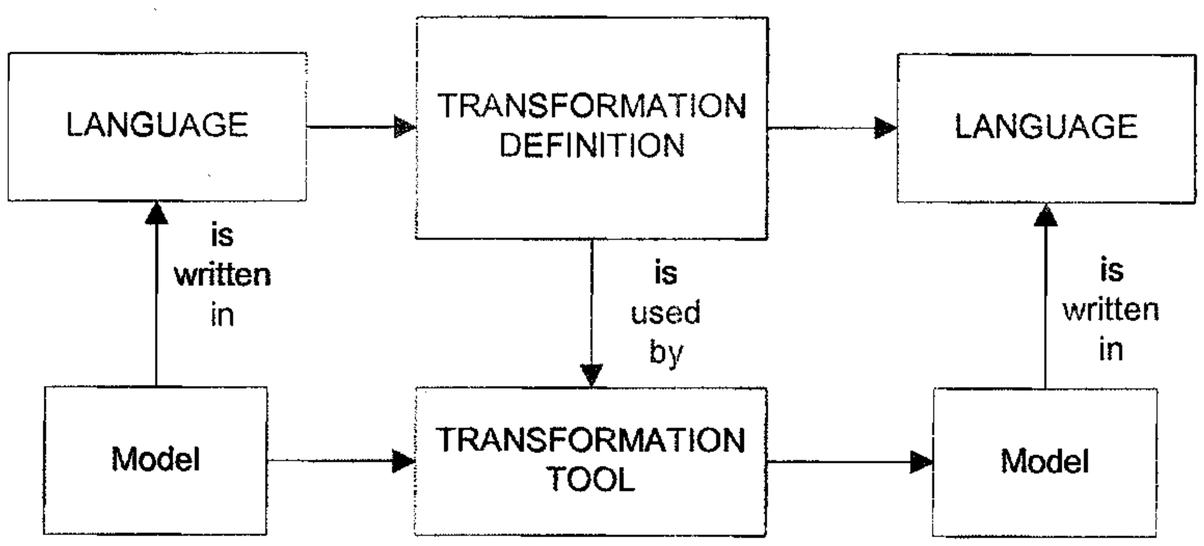
The MDA development lifecycle shown in the preceding figure does not look different from the traditional life cycle. The fundamental difference is in the artifacts created by the process. All the models created in MDA are formal models, i.e. models that can be understood by computers. MDA models can be broken down into the following three major categories:

- **Computational Independent Model (CIM)** is a software independent model used to describe a business system. A CIM may be supported by software, but the CIM itself remains independent of software.
- **Platform Independent Model (PIM)** is a computational model but is independent of the platform specific information of its derived PSMs. Platform independence is a relative term. A model can be independent of the OS, the implementation language, the middleware... Hence, PIMs can be found at multiple levels and can take into account some technical factors.
- **Platform Specific Model (PSM)** is a computational model that depends on the specification of a reference set of platform specific technologies. PSMs are derived from PIMs and take additional technical factors specific to the platform into consideration. In MDA, code is considered to be a Platform Specific Model.

Appendix B.3 discusses MDA models in more details.

Tools can use the information in the models to automate the production of the next level artifact. MDA allows developers to develop a complete application through the refinement of models at different abstraction layers. It is important to note than MDA supports agile processes quite well. The models are simply evolved and refined in an iterative fashion. Because of code generation, the feedback loop is shortened, providing very good feedback to the customers of the application. Because models are now considered to be executable artifacts, they become an integral part of any agile process. Indeed, we can see trends such as agile modeling emerging in the industry.

6.3.2 Transformations



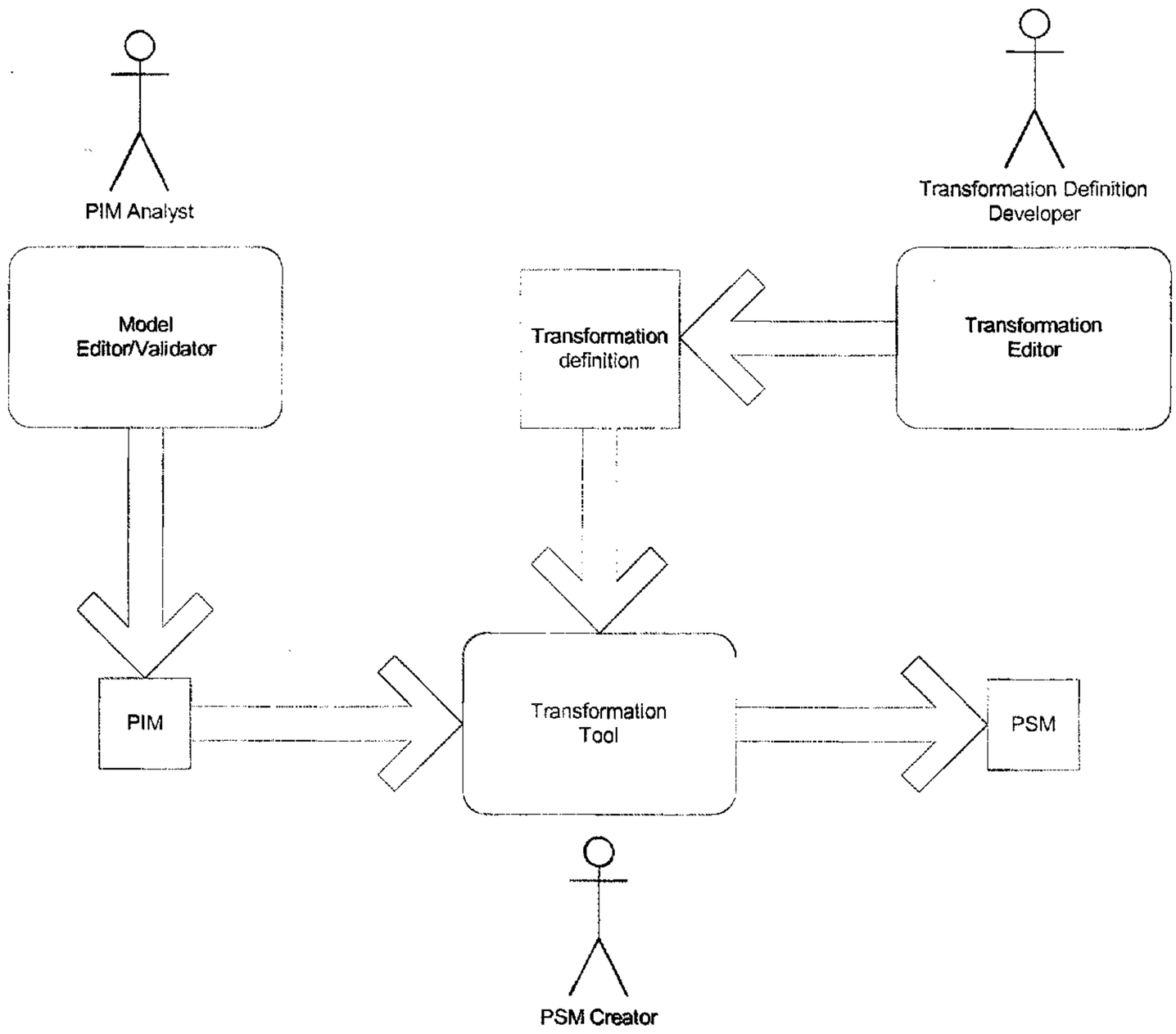
Transformations are essential in the MDA development process. As shown in the preceding diagram, transformations take one model as input and produce a second model as its output. A transformation definition describing how a model should be transformed is used to guide the tool when performing the transformation. As long as both languages have been written in the same meta-language, it is fairly easy to write a proper transformation definition.

In MDA, the possible types of transformations are:

- PIM to PIMs of different levels of abstractions
- PIM to PSMs
- PSM to PSMs of different level of abstractions, including code PSMs

It is also interesting to note that MDA itself does not describe transformations from a Business Model (CIM level) to a PIM. It is strictly concerned with transformations of computational models. Appendix B.2 presents a more complete view of and how transformations fit into the MDA framework.

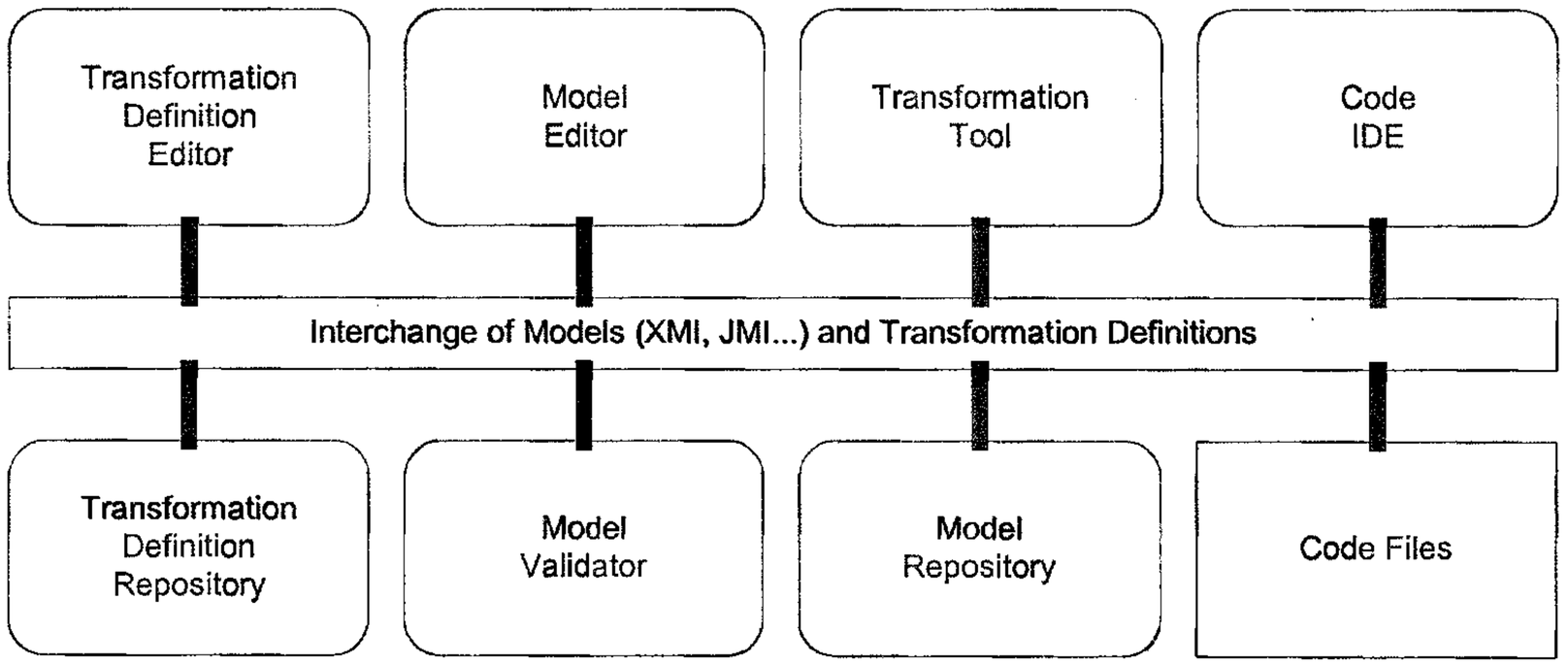
6.3.3 Participants, Tools and Artifacts



6.3.3.1 Participants

In MDA, the **PIM Analyst** is responsible for translating the business models into platform independent models of a system that meets the end-user requirements. Once a PIM of the system is in place and the architecture and the implementation platform is chosen, a **Transformation Definition Developer** will define the mappings required to transform the PIM into a platform specific model. Using transformation definitions, the **PSM Creators** work with a transformation tool refining the models until a fully executable system is produced.

6.3.3.2 Tools



The preceding figure shows the different tools required by MDA:

Transformation Definition Editor: An editor for writing and editing transformation rules

Transformation Definition Repository: The storage for transformation definitions

Model Editor: An editor for creating, editing and viewing the different MDA models

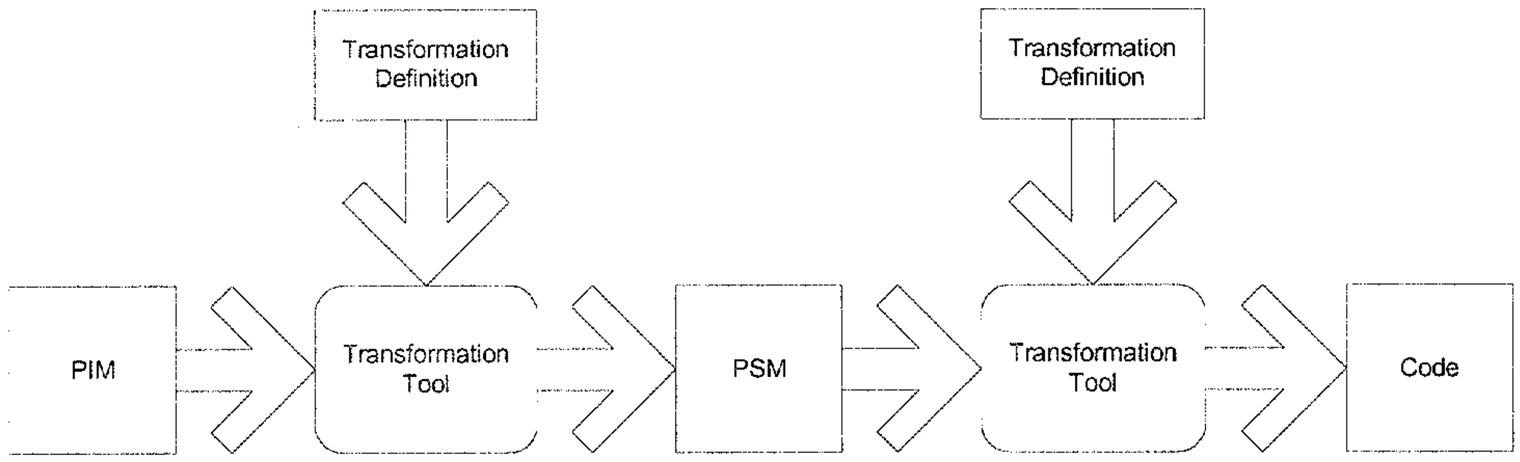
Model Validator: A tool to ensure models contain all the necessary information to make them suitable for transformation

Transformation Tool: A tool that uses transformation definitions to convert a model from one form to another.

Model Repository: The database for all models

Code IDE: The common functions provided by an Interactive Development Environment such as debugging, compilation, and code editing

6.3.3.3 Artifacts



At its most basic, the MDA artifacts are:

Transformation Definitions: A transformation definition consists of a collection of transformation rules, which are unambiguous specifications of the way that one model can be used to create another model.

PIM: Platform Independent Models. A model is said to be a PIM when it abstracts out the particularities of a specified platform.

PSM: Platform Specific Models. A PSM introduces concepts that are specific to a given platform.

Code: Code can be considered to be the lowest level of PSM.

In reality, it is difficult to draw the line between PIM and PSM. The only thing we can say about models is that one model is more or less platform specific that another. In the context of MDA, we always transform a more platform independent model to a model that is more platform specific. Thus the terms PIM and PSM are relative terms.

7 THE SILVER LEAP SOLUTION

7.1 Objectives

This section describes the solution we intend to deliver to facilitate the development of software systems. The high-level objectives of the solution are the following:

- Embrace current industry trends, such as MDA, SOA, BPM and agile processes, which are addressing the most important problems in AD today.
- Facilitate the joint intervention of business owners and technologists ensuring proper alignment of business and IT.
- Simplify development by automating the maximum of required activities and providing adaptive guidance for all skill levels.
- Provide a single, uniform and integrated AD platform supporting the entire lifecycle.
- Facilitate teamwork, reinforcing communication and feedback at all levels.
- Follow standards and allow interoperability with any tool that adheres to them (ex: XMI).

Some of the unique contributions the tool will bring include:

- Bridge the gap between BPM and System development.
- Provide adaptive process guidance with an emphasis on helping AD generalists deliver complete solutions.
- Provide simple default MDA transformations that can be customized and adapted instead of requiring a lot of upfront investment before being able to use the tool effectively.
- Support for agile modeling vs. traditional Big Design Up Front (BDUF).
- Support test-driven development, integrating tests into the entire lifecycle.
- Communicate accurately and at all times the project status to the whole team.

7.2 Solution

The solution proposed by Silver Leap is the ***ACES Integrated Environment***. This *Model and Develop* environment consists of a suite of tools that guide the user through an agile and lightweight ARAD like process. The process has the particularity of being based on highly formalized models and providing a substantial amount of guidance in order to allow the automation of the vast majority of the activities. The environment is ideal for iterative agile processes such as XP, Scrum, the unified ARAD process (based on RUP) and others.

7.3 Audience

The ***ACES Integrated Environment*** is primarily targeted at corporate developers involved in the development of large J2EE enterprise applications. Based on the current failure rates for projects of this type, we strongly believe that the current set of tools is simply insufficient to meet their real needs. Certainly, it is quite evident that current tools require a very specialized level of knowledge that is increasingly difficult to find as applications and frameworks get more and more complex with time.

The tool is targeted first at software generalists who can build complete applications without being experts in everything. Specialists can then be used to refine and improve the solution as required. Because it supports the entire development lifecycle, the tool addresses all AD disciplines.

7.4 Addressing the problems

7.4.1 Complexity

At the core of the ACES Integrated Environment is an MDA engine. The MDA engine raises the level of abstraction so engineers can deal with more complex problems without being entangled in low-level details. The tool also provides a comprehensive set of wizards that guide the users through the different MDA transformations. Default MDA transformations provide the basic functionality to produce a running system based on the user input models at the CIM and PIM levels.

7.4.2 Business/IT alignment

The ACES Integrated Environment fully integrates business modeling into the development process; it ensures developers build solutions that truly originate

from clearly identified business needs. The integration of automated acceptance testing practices and tools in the process helps ensure that what the developers build does indeed meet the user requirements. The tool also supports agile project planning which puts emphasis on providing quick and real feedback to the entire team.

7.4.3 Changing Business Needs

The ACES Integrated Environment is an MDA based Architected Rapid Application Development (ARAD) tool. The high level of automation and the agility provided by practices such as TDD (Test Driven Development) make it possible to create a highly responsive development organization.

7.4.4 Quality

By integrating testing activities during all the AD lifecycle, the ACES Integrated Environment provides a development environment where quality is not just an afterthought. Quality is built into the process and helps support agility. Furthermore, the automation of most of the executable artifacts substantially raises the level of quality.

7.4.5 Technological innovation

The pace of technological innovation very often causes the same applications to have to be redone every couple of years. The Web and mobile platforms are good examples of some of the most recent technological innovations. Even though business processes and logic are mostly technology independent, they are often embedded tightly into technology dependent containers that can't be reused easily. By merging BPM technology and MDA, the ACES Integrated Environment ensures that the investment in systems will not be completely lost as new technology rolls in. Investment in business models and platform independent models can be reused easily over time.

7.4.6 Architectural Integrity

Large projects often suffer from a lack of architectural integrity. Systems architects using the ACES Integrated Environment can easily ensure architectural integrity by influencing the design of the models at the appropriate levels. These high level models represent guidelines that engineers then refine to eventually produce a running system.

7.4.7 Application Integration

Systems rarely work in a vacuum; most rely on other external systems and information. By incorporating SOA technologies into its offering, Silver Leap makes it easier for engineers to connect existing systems to other systems as well as to expose and package the functionality of new systems as services. These services can than be used by other systems or organizations. Integration

is also made easier by having the ability to assemble different pieces of functionality at the model level.

7.4.8 Reuse

Over the years, even with the introduction of object-orientation, re-use at the level of code has proven to be rather elusive. The ACES Integrated Environment supports reuse by facilitating the use and definition of web services. It also provides multiple assistants and wizards to help engineers reuse platform independent models and patterns (business, analysis and design).

7.4.9 MDA limitations

The ACES Integrated Environment fully supports the MDA initiative and tries to compensate for the limitations of this important trend. In particular, the modeling language used for MDA is incomplete. In the meantime, our solution will offer a language that offers the following qualities:

- Expressive enough to specify the systems completely
- Non application specific language
- Provide higher level constructs
- Suitable for n-tier development
- Suitable for distributed applications
- Seamlessness between models and applications
- Support for managing large models

For a more extensive description of MDA limitations, consult Appendix B.4.

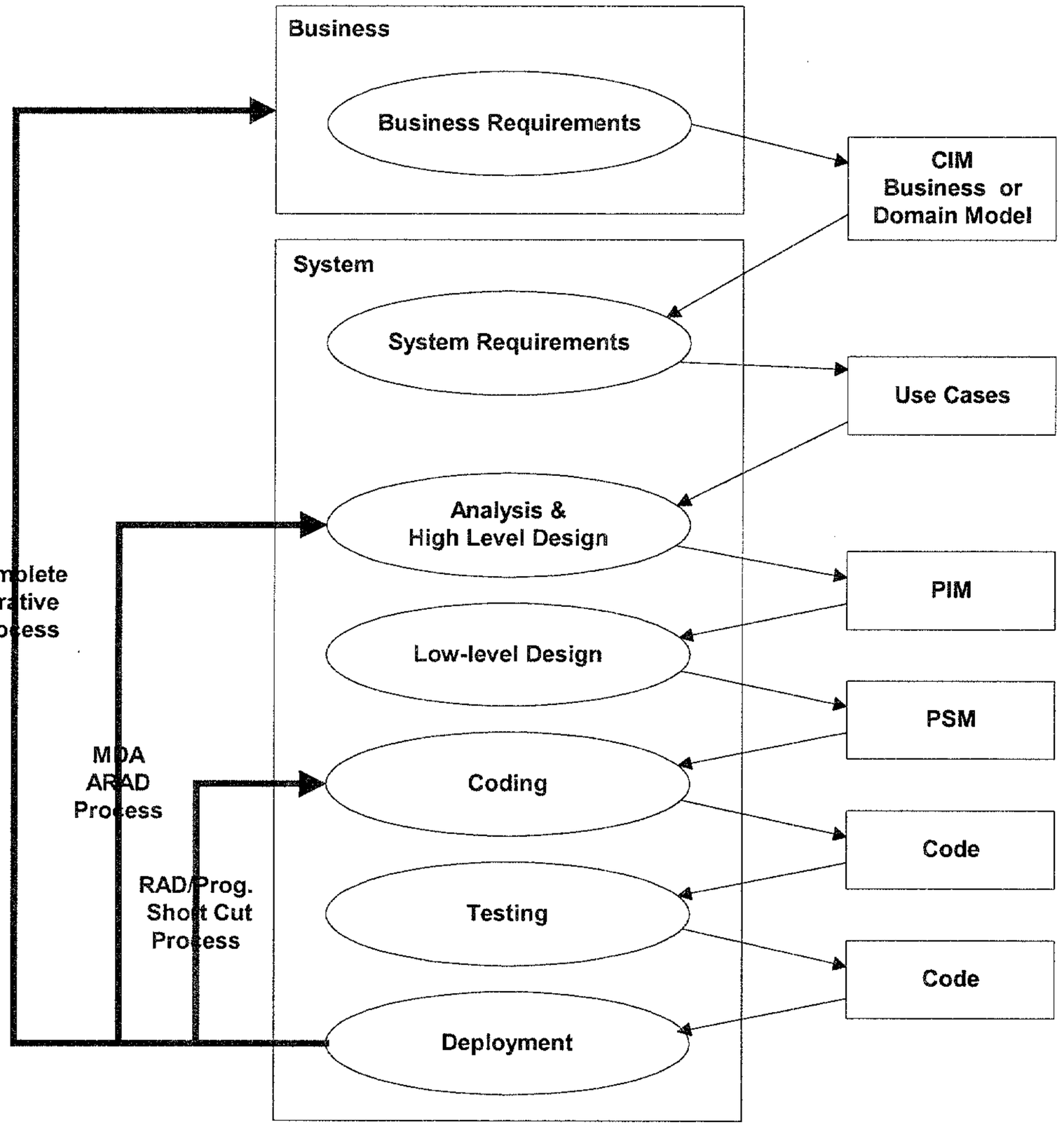
7.4.10 Integrated Environment

Although it is possible to pick and choose different best of breed development tools to achieve a complete ALM solution, it is an extremely hard and frustrating task. Furthermore, there is rarely a good synergy amongst tools that were not made to work together and/or just patched up to achieve marketing gains. The current trend in AD is for vendors to offer a completely integrated solution. These tools usually work well together and offer a level of simplicity and productivity that is not attainable otherwise. The ACES Integrated Environment is a complete suite of tools that encompasses the entire application development lifecycle. It provides tools for modeling the business and the application, transforming models into code, building and deploying the application as well as managing all aspects of development.

All the tools provided in the ACES Integrated Environment are built on top of Sun's NetBeans IDE Platform. The NetBeans Java module is provided to offer the necessary Java development facilities.

7.5 Process Overview

The ACES Integrated Environment supports a model-driven and refinement based approach. Emphasis is put on the iterative aspects of a software development, a very strong business-based focus and the reliance on tools to enforce strict transformation rules.



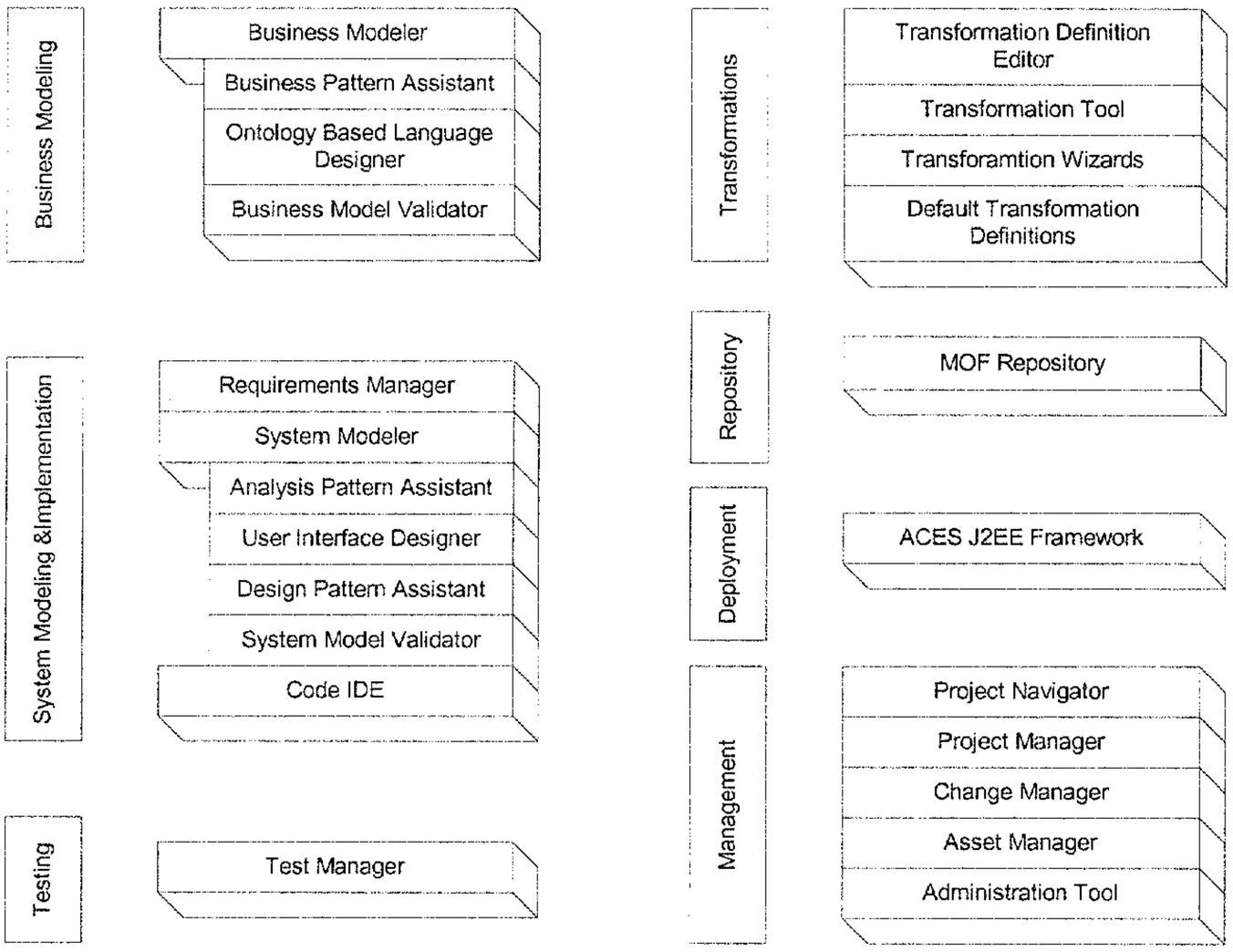
The preceding figures shows how the proposed process differs from typically used MDA/ARAD as well as RAD/traditional type processes. The iterative feedback loop has been extended to include the business modeling and requirements management activities. The net effect of closing that loop is to continuously allow feedback of all the stakeholders to insure the team is not only building the software the right way but that they are also building the right thing.

To complete the ALM offering, the ACES Integrated Environment provides integrated Configuration & Change Management as well as Project Management tools (which are not shown in the figure).

At its core, the process is one of transformation and refinement. Starting from the business model at the highest, transformations that embody architectural and design decisions are applied to increasingly detailed models. At the lowest level we find a code model (code PSM) that can be compiled into an executable system. Default sets of parameterized transformations are also provided to facilitate the production of a complete n-tier application from a PIM.

7.6 Features

7.6.1 Tools Overview



7.6.1.1 Business Modeling Tools

Business Modeler: used to assist the user through the business modeling steps.

Business Patterns Assistant: used by the Business Modeler to assist the business analyst in applying known business patterns to a model. The users can add new patterns in the pattern repository as they are discovered.

Ontology Based Language Designer: used by the business analysts and the software architects to define a business-specific language that will be used for the creation of all business models.

Business Model Validator: used to validate that the business model respects the well-formed rules required for the tool to be able to do the appropriate transformations.

7.6.1.2 System Modeling & Implementation Tools

Requirements Manager: used to establish the system requirements based on the business requirements.

System Modeler: used to guide the software architect to build and refine software models from Platform Independent Models down to Platform Specific Models at different levels of abstraction. It is basically a UML modeling tool supporting the development of static and dynamic models of the application.

Analysis Patterns Assistant: used by the System Modeler to assist the software architect into applying analysis patterns to the software models. The tool allows the users to add new patterns as they are discovered.

User Interface Designer: used to quickly develop the system's user interface at different abstraction levels.

Design Patterns Assistant: used by the tool to assist the software programmer into applying design patterns to the various PSMs. The tool allows the users to add new patterns as they are discovered.

System Model Validator: used to validate that the system model respects the well-formed rules required for the tool to be able to do the appropriate transformations.

Code IDE: provides support for code development activities such as debugging, compilation, code editing, unit testing and deployment.

7.6.1.3 Testing Tools

Test Manager: provides control, management and reporting of all test activities. It supports functional test automation and traceability with system requirements.

7.6.1.4 Transformation Tools

Transformation Definition Editor: This editor allows the user to specify what transformation rules will be applied to go from one type of model to another. The transformations definitions are expressed using a language described in the Query, Views and Transformations (QVT) Standard and are stored in the MOF repository.

Transformation Tool: used to transform a model into another according to the rules specified in a given transformation definition. A tuning wizard must allow the parameterization of the transformation process in order to apply particular settings.

Transformation Wizards: used with the transformation definition editor to assist in defining the different model transformations. The transformations wizards are available at different levels, from business models to system PIMs, from PIMs to PSMs and from PSMs to code.

Default Transformation Definitions: a default set of transformation definitions that support the generation of web transactional N-tier architecture based applications running on the J2EE platform.

7.6.1.5 Repository Tools

MOF Repository: used to store the information about the models and other project artifacts.

7.6.1.6 Deployment Tools

ACES J2EE Framework: the default J2EE transformations provided with the tool target a framework that encapsulates the J2EE best practices and allow generated applications to be more maintainable. Alternatively, transformations can be done to go directly to the J2EE framework.

7.6.1.7 Management Tools

Project Navigator: is the primary interface integration point of the tool. It manages the interventions of the various users by allowing them to dig into the models according to their expertise and level of understanding of the project.

Project Manager: provides project managers with the ability to track and manage project scope and assess and report on project progress.

Change Manager: provides defect and change tracking.

Asset Manager: provides version control and build management.

Administration Tool: allows the global administration of the tool settings.

7.6.2 Using the ACES *Integrated Environment*

The following section details how the different tools of the *ACES Integrated Environment* can fulfill its objectives.

7.6.2.1 Business Modeling

7.6.2.1.1 Guide the business analyst through business modeling

The steps required to model a business are numerous and difficult. The methods and notations traditionally used are also diverse and more or less formal. The *Business Modeler* will guide the business analysts through the business modeling steps by guiding him or her through the different steps in the right order and using the appropriate notation. Wizards will let the user enter information iteratively.

The project navigator tool will allow the users to return to the modeling activities at various levels of abstraction and complexity depending on their level of intervention.

The methodology chosen for modeling the business is object-orientation and the notation used for this activity is UML. Activity diagrams are used for business processes (assembly lines), business use cases for system functionalities, object and class diagrams for business entities. The creation of business use cases is formalized to allow the platform to extract valuable information from them. Those diagrams essentially drive all subsequent activities.

7.6.2.1.2 Enforce formalism for business modeling

One of the key constraints that should be respected when doing business modeling is the tight integration between business models and supporting software models. We saw previously that this tight integration and the constant synchronization of both present many advantages. For this to happen, many factors must be in place as described in the section "From business models to software architecture". First, the notation used must preferably be the same in the two cases. Second, the set of modeling diagrams used must be restrained to a set of "formal enough" diagrams. Third, all the artifacts must use a language that is configurable to the current problem but generic enough to be interpreted by processors (ex. transformation engines).

For the *ACES Integrated Environment*, UML is chosen as the default business modeling notation. Second, we restrain the use of the UML diagrams to Activity, Object, Class, Statechart, Business Use Case and Package diagrams. The diagrams are all stored into the *MOF Repository*. Different wizards guide the elaboration of the models and the *Business Model Validator* allows the constant verification of all the produced artifacts.

The *Ontology-based Language Definer* allows the business analysts to define their own language and use it thereafter for the business modeling activities. All the entities defined (be they text or resources) will be stored into the *MOF Repository* and perfectly usable for the subsequent steps. Business analysts can

then use other standard BM notations (such as BPML) instead of UML. This ability fulfills one of the key objectives of the solution.

7.6.2.1.3 Allow the business analyst to use efficiently the business patterns

Business patterns can bring tremendous value to a project if applied correctly. The *ACES Integrated Environment* supports the insertion of business patterns from a catalog by simply selecting one and personalizing it. The *Business Patterns Assistant* determines the applicability of the pattern. Wizards ask the user to specify the information necessary to personalize the pattern to the current needs.

7.6.2.1.4 Provide extensive wizards to assist the analysts in translating business models into system requirements and specifications

The section "From a Business model to a Software model" illustrates the numerous steps that must be taken to achieve this conversion. Process diagrams (assembly lines) are first used to determine what part of the business has to be automated. From there, the tool assists the business analysts in creating business use cases. The resources defined in the class and object diagrams of the business models are then used as the basis for the static diagrams needed in the system models. Patterns exist in Responsibility Driven Design [GRAHAM 1997] to effectively map together these kinds of elements. Appendix A.5 illustrates how this conversion can be made.

Entities that have not been identified during the business modeling activities or represent variations of these can be added to the system models. These newly created resources must be added to the repository and as such, become available among the other business resources. This crucial activity is done in an iterative manner and allows the progressive refinement of the software models and business models while always keeping them synchronized.

7.6.2.2 System Analysis, Design and Implementation

7.6.2.2.1 Guide the software analyst through the complex steps of software analysis and design.

The tool enforces the process by guiding the developers through the necessary steps and making sure that the appropriate artifacts are produced. The *System Model Validator* verifies that system models respect the notation and standards prescribed in order to maintain their "executability". Iteratively, the steps based wizard allows the users to complete the necessary activities in a visual and user-friendly fashion.

7.6.2.2.2 Allow the software analysts to use efficiently the available analysis patterns

This functionality is the same as the one describing the business patterns except that it applies in the context of a system models instead of business models.

7.6.2.2.3 Model the application

The *System Modeler* can create, modify and display the different system models. UML is also the notation used for system models. The *System Modeler* provides a way to show the different abstraction levels of the models and the transformations rules that have been applied between them.

7.6.2.2.4 Create and manage transformation definitions

The *ACES Integrated Environment* provides tools that follow the concept of model transformations. The transformations incorporate technical and architectural decisions at different levels. For example, design patterns are implemented via transformations. Using the *Transformation Definition Editor,* a user can specify which transformation rules will be applied to go from one type of model to another. The transformations definitions are expressed using a language described in the Query, Views and Transformations (QVT) Standard and are stored in the *MOF Repository*.

7.6.2.2.5 Execute model transformations

The *ACES Integrated Environment* provides a *Transformation Tool* that automatically generates target models from source models according to the rules provided in transformation definitions. A tuning wizard is provided to allow the parameterization of the transformation process in order to apply particular settings.

7.6.2.2.6 Provide default transformation definitions for the main middleware platforms

We have seen in the previous sections that MDA mandates the use of PIM and PSMs to make the link between a conceptual model and a specific implementation. We consider that the ability to create these transformations (which could become a tremendous undertaking) is not enough to provide a really interesting solution to software designers. Using the best practices and standards as much as possible, the *ACES Integrated Environment* provides default transformations for the J2EE platform. These transformations can be used as is or customized by the architects to suit their specific needs. The customization can be done using the *Transformation Definitions Editor* and/or the *Transformation Wizards*.

7.6.2.2.7 Design user interfaces

The design of user interfaces is a delicate part of any software project. They are often different from project to project and the automation of their production is always difficult. When designing user interfaces, many different aspects like ergonomics, technology projection (Web, desktop, WML, etc.) and efficiency are intermixed. They are seldom included in the modeling phase but are instead taken as a side project.

The *User Interface Designer* provides a platform independent user interface designer. It allows the user to create quickly a model of the user interface. A number of transformation definitions are provided to support the generation of the

user interfaces on different platforms. Default transformation definitions can easily be extended and modified to support a customized look and feel.

The *User Interface Designer* allows the visual design of various and complex user interfaces and the user interfaces are represented as an integral part of the model.

7.6.2.3 Management

7.6.2.3.1 Navigate the project at different levels of abstraction

Development of an application using the *ACES Integrated Environment* consists essentially of creating models and applying transformations. The *Project Navigator* allows the user to navigate through the different abstraction layers. It shows the refinement structure of the models and the transformations that are applied between them. All the other tools can be called from the navigator.

7.6.2.3.2 Manage projects

The *ACES Integrated Environment* provides a configurable project management facility supporting iterative processes.

The main project management functions include:

- Release and iteration planning (Tasks, Time & Resources)
- Project Tracking
- Project Dashboard (Status Reporting) featuring direct hooks to the *Test Manager* for traceability

The tool provides configurations for agile processes like XP and Scrum as well as a RUP ARAD configuration.

7.6.2.3.3 Manage software assets

The *Asset Manager* supports versioning of all development artifacts.

7.6.2.3.4 Track changes and defects

The *Change Manager* manages change requests and bugs.

7.6.2.4 Testing

7.6.2.4.1 Integrate testing into the process

Testing has always been a critical piece of the software development lifecycle but is even more critical for practitioners of iterative and agile processes. The *ACES Integrated Environment* ups the ante by supporting the concept of Test Driven Development. In TDD, automated tests drive the development at all levels (system or acceptance, integration and unit). Tests are written before any modification is made to the system.

The *Test Manager* provides support for managing all tests and for reporting test results. Unit testing is provided by the inclusion of the JUnit framework. Acceptance testing is provided via the Fit framework. Whenever possible, the environment provides the design and automatic generation of tests. For example, unit tests can be generated from OCL constraints defined in UML models.

7.6.2.5 Miscellaneous

7.6.2.5.1 Facilitate the integration of legacy and enterprise systems

The *ACES Integrated Environment* provides the necessary transformation definitions to facilitate the creation of a Services layer according to the SOA (Service Oriented Architecture). As described previously, this architecture focuses on the creation of decoupled services accessible and discoverable over a network. These services are mostly implemented as web services that the transformations must support.

7.6.2.5.2 Integrate with commonly used development tools

Realizing the value of some of the contributions of the open source community, The *ACES Integrated Environment* takes an extra step to integrate them into a coherent and efficient whole. Some of the tools integrated in the suite include: NetBeans, MDR, CVS, Ant, Struts, JUnit and Cactus.

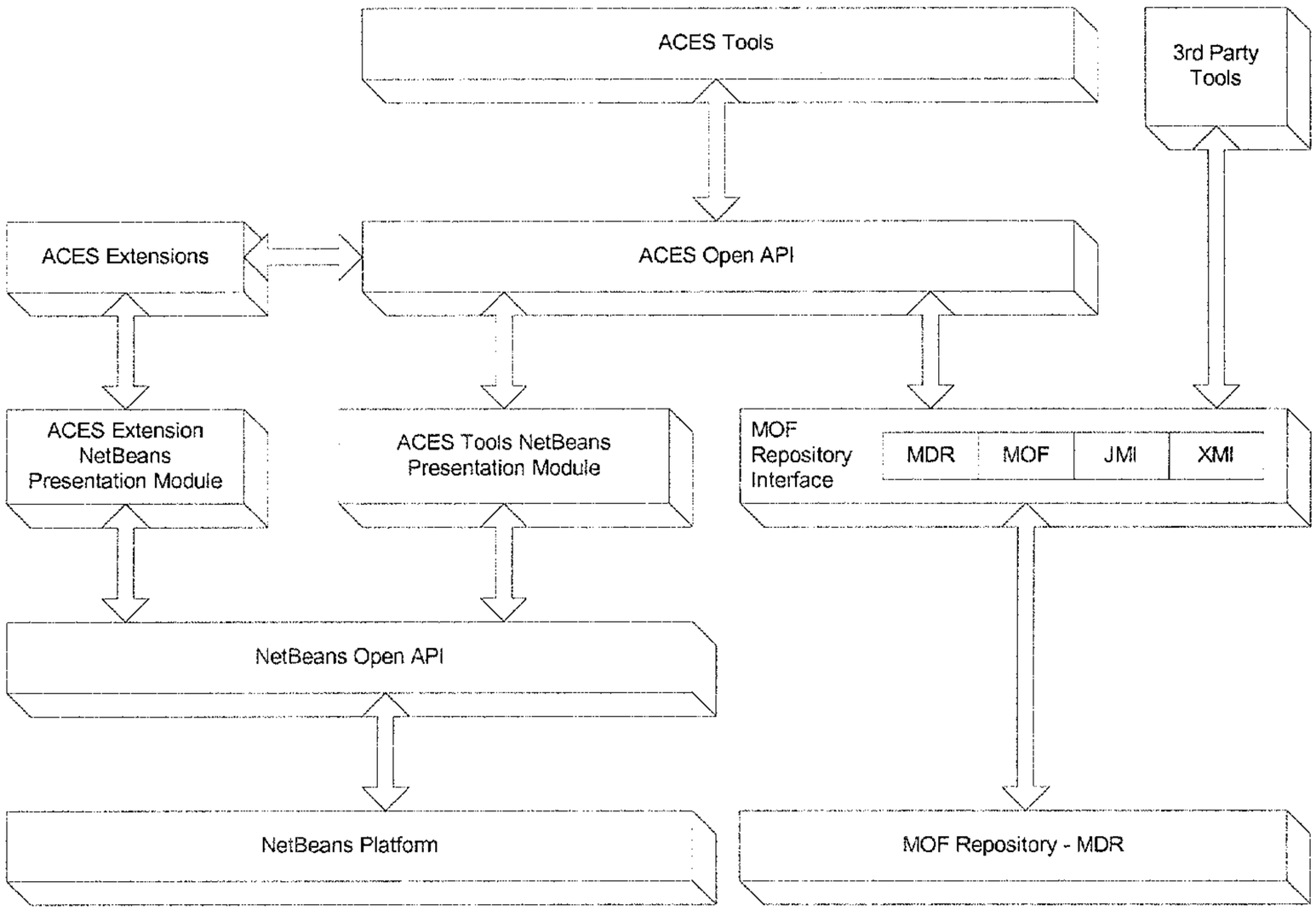
7.6.2.5.3 Support for MDA Standards

The *ACES Integrated Environment* is an environment built on open standards, allowing other tools to be inserted in the process as needed.

The underlying repository that stores all project artifacts is based on OMG's MOF specification and supports XMI import/export. UML, the most widely used modeling language is used for all modeling activities in the environment. Appendix B lists the MDA related standards that is supported.

7.6.3 Physical architecture

7.6.3.1 Development Environment



7.6.3.1.1 Overview

The *ACES Integrated Environment* is based on Sun's Open Source *NetBeans Platform*, Sun's open IDE toolkit written in Java. It can be used to deliver useful developer tools by providing basic core services and infrastructure such as windowing, actions and file management. In the context of *ACES*, the *NetBeans Platform* provides the primary integration point for all the *ACES tools*.

The *ACES NetBeans Presentation Module* is the NetBeans presentation layer of the ACES tools, integrating the user interfaces of all the ACES tools into the *NetBeans Platform* through the *NetBeans Open API*. The *NetBeans Platform* provides the *NetBeans Open API* to allow developers to add functionality to the base version of the platform by providing custom modules.

The *ACES Open API* abstracts the functionality of all the *ACES tools*. This API is used by the *ACES NetBeans Presentation Module* to connect the presentation layer to the ACES functionality. This API is also used by *ACES* extensions to extend or supplement existing ACES functionality. An *ACES Extension NetBeans Presentation Module* must be provided to expose the user interface of ACES Extensions if needed.

Sun's *Metadata Repository* or *MDR* is an implementation of a *MOF repository*. It is used to store all the models and other information used by the ACES Integrated Platform.

Access to the *MOF repository* is provided via a number of possible interfaces:

- **MDR**: the MDR API defines a generic API (except the standard API defined in JMI) for metadata repositories. This API is an extension to JMI API (adding API for events, facility, etc.)
- **MOF**: Library containing generated JMI interfaces for MOF 1.4.
- **JMI**: the JMI API contains JMI reflective interfaces specified by JSR-40 (JMI).
- **XMI:** The MDR provides a library containing generic JMI utilities that operate on JMI reflective interfaces. These include implementation of an XMI reader and XMI writer.

Most ACES tools interact with the repository through JMI. However, *3rd Party Tools* that support XMI can easily exchange artifacts with the different *ACES Tools*. *3rd Party Tools* providing a more advanced plug-in architecture can easily be integrated to ACES by connecting through the *ACES Open API* in a way similar to the *ACES Extensions*.

7.6.3.2 Deployment Environment

7.6.3.2.1 Application Deployment

The *ACES Integrated Environment* produces J2EE based applications.

The preferred deployment environment for applications generated by this tool is the SunONE family of middleware servers (web, portal, application). But support is provided for all J2EE compliant middleware.

Persistence services are available using any JDBC compliant database. MySQL is provided as the default database.

Common N-Tier application patterns and best practices are encapsulated in the *ACES J2EE framework*. Applications generated with the default transformations will make use of this framework.

All configuration-compiler-deployment issues are managed with ANT. This open-source technology is the de-facto standard for Java based deployments. The tool automates the creation of the necessary configuration files.

7.6.3.2.2 Aces Deployment

ACES is a Java 2 application based on NetBeans. It requires a 1.3 (or higher) JRE and JDK. It can run on Linux, Solaris and Windows.

8 CONCLUSION

The *ACES Integrated Environment* suite of tools will represent a major milestone in the application development industry with its very high level of integration of the various steps and artifacts necessary to successfully achieve business modeling and supporting software development. It is highly MDA based but provides more functionality in the Business Modeling field and compensates for the current weaknesses of MDA.

Appendix A: Business Process Modeling

A.1 A primer on OCL

The Object Constraint Language (OCL) is a language that enables one to describe expressions and constraints on object-oriented models and other object modeling artefacts. An expression is an indication or specification of a value. A constraint is a restriction on one or more values of (part of) an object-oriented model or system. Various constraint languages have been used in object-oriented modeling methods (Syntropy, Catalysis, and BON), and programming languages (Eiffel). The OCL is a standard query language, part of the Unified Modeling Language (UML), the Object Management Group (OMG) standard for object-oriented analysis and design.

There are four types of constraints:

An ***invariant*** is a constraint that states a condition that must always be met by all instances of the class, type, or interface. An invariant is described using an expression that evaluates to true if the invariant is met. Invariants must be true all the time.

A ***precondition*** to an operation is a restriction that must be true at the moment that the operation is going to be executed. The obligations are specified by post-conditions.

A ***post-condition*** to an operation is a restriction that must be true at the moment that the operation has just ended its execution.

A ***guard*** is a constraint that must be true before a state transition fires.

Context of a constraint

Constraints are restrictions on a model or system, they are always coupled to the items used in that model, usually a series of UML diagrams. Actually, the OCL can be used in any model as long as it supports the basic notions of class and instance, attributes, associations and operations.

Each constraint is linked to one item from the model. This item is called the context of the constraint. If the constraint is an invariant, the context is a class. If the constraint is a pre- or post-condition, the context is an operation of a class. If the constraint is a guard, the context is the state from which the transition fires. Note that in all cases, the keyword self refers to the instance of that class for which the constraint is being evaluated, e.g. the class that defines the operation, or the class for which the state chart is specified.

Invariants on attributes

The simplest constraint is an invariant on an attribute. Suppose our model contains a Customer class with an attribute age, the following constraint restricts the value of the attribute:

context Customer inv: age >= 18

Invariants on associations

One may also put constraints on associated objects. Suppose our model contains the Customer class. This class has an association to the Salesperson class, with role name salesrep and multiplicity of 1. The following constraint restricts the value of the attribute knowledge level of the associated instance of Salesperson:

context Customer inv: salesrep.knowledgelevel >= 5

Collections of objects

In most of the cases the multiplicity of an association is not 1, but more than 1. Evaluating a constraint in these cases will result in a collection of instances of the associated class. Constraints can be put on either the collection itself, e.g. limiting the size, or on the elements of the collection. Suppose in our model the association between Salesperson and Customer has role name clients and multiplicity 1..* on the side of the Customer class, then we might restrict this relationship by the following constraint.

context Salesperson inv: clients->size() <= 100 and clients->forAll(c: Customer | c.age >= 40)

Pre- and post conditions

In pre and post-conditions the parameters of the operation may be used. Furthermore, there is a special keyword result that denotes the return value of the operation. It can be used in the post condition only. As an example we have added an operation sell to the Salesperson class.

context Salesperson::sell(item: Thing): Real pre: self.sellableItems->includes(item) post: not self.sellableItems->includes(item) and result = item.price

Broken constraints

Note that evaluating a constraint does not change any values in the system. A constraint states, "this should be so". If for a certain object the constraint is not true (in other words, it is broken), the only thing we can conclude is that the object is not correct; it does not conform to our specification. Whether this is a fatal error or a minor mistake, and what should be done to correct the situation is not expressed in the OCL.

A.2 Introduction to ASL - Action Semantics Language.

The aim of the language is to provide an unambiguous, concise and readable definition of the processing to be carried out by an object-oriented system. The ASL definition is independent of any particular implementation. The ASL language provides the following:

- Sequential Logic
- Access to the data described by the Class Diagram
- Access to the data supplied by signals initiating actions
- The ability to generate signals
- Access to timers
- Access to synchronous operations provided by classes and objects
- Access to operations provided by other domains
- Tests and Transformations

ASL is organized into "segments" each of which is a sequential set of ASL statements with a local data scope. In xUML, ASL segments can be used to define:

- The processing to be carried out on entry to a state
- The processing to be carried out by a method behind an operation
- The start-up sequence for a system either for test or for target system purposes
- The processing to be carried out by test methods for simulation purposes
- The processing to be executed in bridges providing the mapping between domains

The execution rules for a segment are as follows:

- Execution commences at the first ASL statement in segments and proceeds sequentially through the succeeding lines as directed by the logic structures
- Execution of the segment terminates when the last ASL statement is completed
- The only external data available to the ASL segment is:
- Signal data supplied with the signal or operation call that caused execution of the ASL segment
- Attributes of classes as defined in the Class Diagram
- There are no "global" data other than those detailed in the Class Diagram

- Local variables created within the segment go out of scope when execution of the segment ends

Critics of AS state that it is not abstract enough to be used successfully as a general purpose MDA action language. Some of its weaknesses may come from the fact that Executable UML, which introduced AS, was mostly addressing MDA concerns in the limited context of real-time embedded systems.

Mapping between Model and Implementation

The xUML model is stored into a repository. And the data inside the repository is analyzed with a tool for completeness and correctness. The constraints defined into the Class Diagram are written to an XML file. The segments (defined inside the State Machine using the State Chart Diagram) are also written to an XML file. The model compiler then combines these files to generate the plat-form dependant code. During this process many aspect of the solution can be selected to generate the pertaining code depending on the requirements of the system and/or the environment.

A.3 BPM Related Standards

BPML: The Business Process Modeling Language (BPML) is a meta-language for the modeling of business processes, just as XML is a meta-language for the modeling of business data. BPML provides an abstracted execution model for collaborative & transactional business processes based on the concept of a transactional finite-state machine.

BPEL4WS: A business process and choreography API that was co -authored by IBM, Microsoft and BEA. It has been formally submitted to the new OASIS Technical Committee at its first meeting on 16 May 2003. The new OASIS Web Services Business Process Execution Language (WSBPEL) Technical Committee will continue work on the Business Process Execution Language for Web Services (BPEL4WS) specification.

EbXML: The Electronic Business using eXtensible Markup Language, sponsored by UN/CEFACT and OASIS, is a modular suite of specifications that enables enterprises of any size and in any geographical location to conduct business over the Internet. Using ebXML, companies now have a standard method to exchange business messages, conduct trading relationships, communicate data in common terms and define and register business processes.

BPMN: The Business Process Modeling Notation (BPMN) specification provides a graphical notation for expressing business processes in a Business Process Diagram (BPD). The BPMN specification also provides a binding between the notation's graphical elements and the constructs of block-structured process execution languages, including BPML and BPEL4WS. The first draft of BPMN was made available to the public on November 13, 2002.

BPQL: The Business Process Query Language defines a standard interface to the forthcoming Business Process Management Systems (BPMS). It allows

system administrators to manage the BPMS and business analysts to query the instances of business processes it executes.

A.4 Different views to describe a business

A.4.1 Business Vision view

The business vision view depicts the company's goals. It is an image of where the company is headed and sets up the overall strategy for the business. It acts as a guide for modeling the other views. According to Goeffrey Darnton [DARN 1997], there are important factors to consider when creating the Business Vision view. These are as follows:

- Company mission
- Objectives
- Strengths
- Weaknesses
- Opportunities: areas of potential growth for the business
- Threats: external conditions that might negatively affect the business
- Critical factors: elements that are required for the business to succeed
- Strategies: action plans to achieve the objectives
- Core competencies
- Roles: the specific functions of the people working in the business
- Organization units: groups into which the business is divided
- Key processes

Usually, a member of the upper management, such as the CEO, is responsible for creating this view. Many techniques exist to create this view but what is important is that this information exists in order to keep the vision of the business in front of all the other modeling activities. We will see later that the goals expressed in the vision view will be used in the process modeling to express the fact that a process is used to satisfy one or more of the company's goals. In conclusion, what is important is that the goals or objectives of the business be expressed in a way granular enough to be allocated to single business processes.

A.4.2 Business Process view

The Business Process view is at the center of business modeling. The processes show the activities that must be undertaken to achieve an explicit goal, along with their relationships with the resources participating in the process. The result of

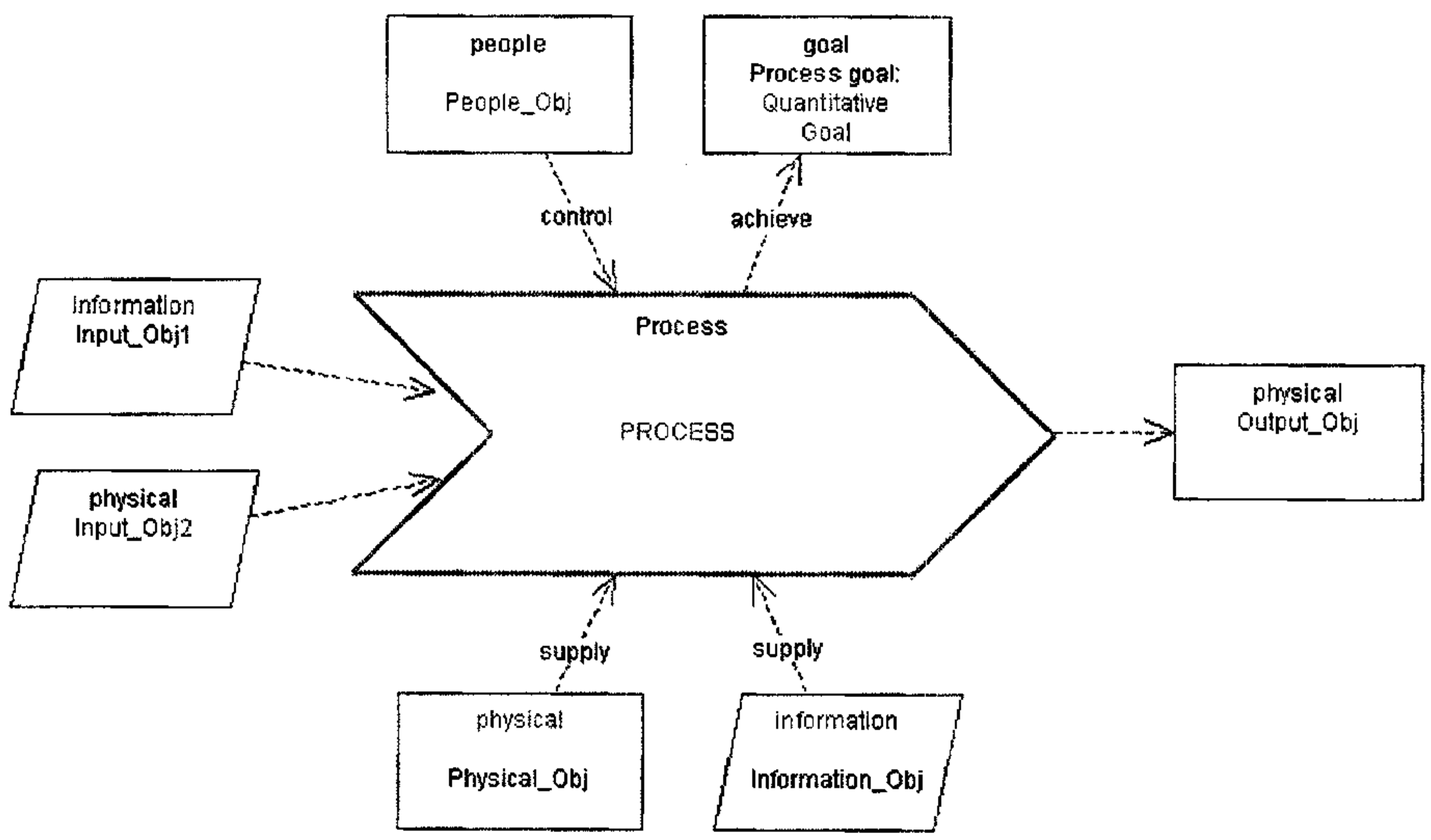
this modeling phase is a number of process diagrams that describe at least the core processes of the company. Core processes are normally customer-oriented and horizontal to the traditional organization of the company. The Business Process view is described with a UML activity diagram.

Processes are modeled first by concentrating on and completely modeling the core processes of the business and then moving on to the support processes. A business usually does not have more that five to ten core processes but may have many more support processes.

Process modeling creates an accurate documentation of the way in which the work is performed, perhaps in order to develop better information systems. It is also used to improve or innovate processes.

Another area of interest related to the process modeling is the process management that allows the tuning, evolution and support of the business processes.

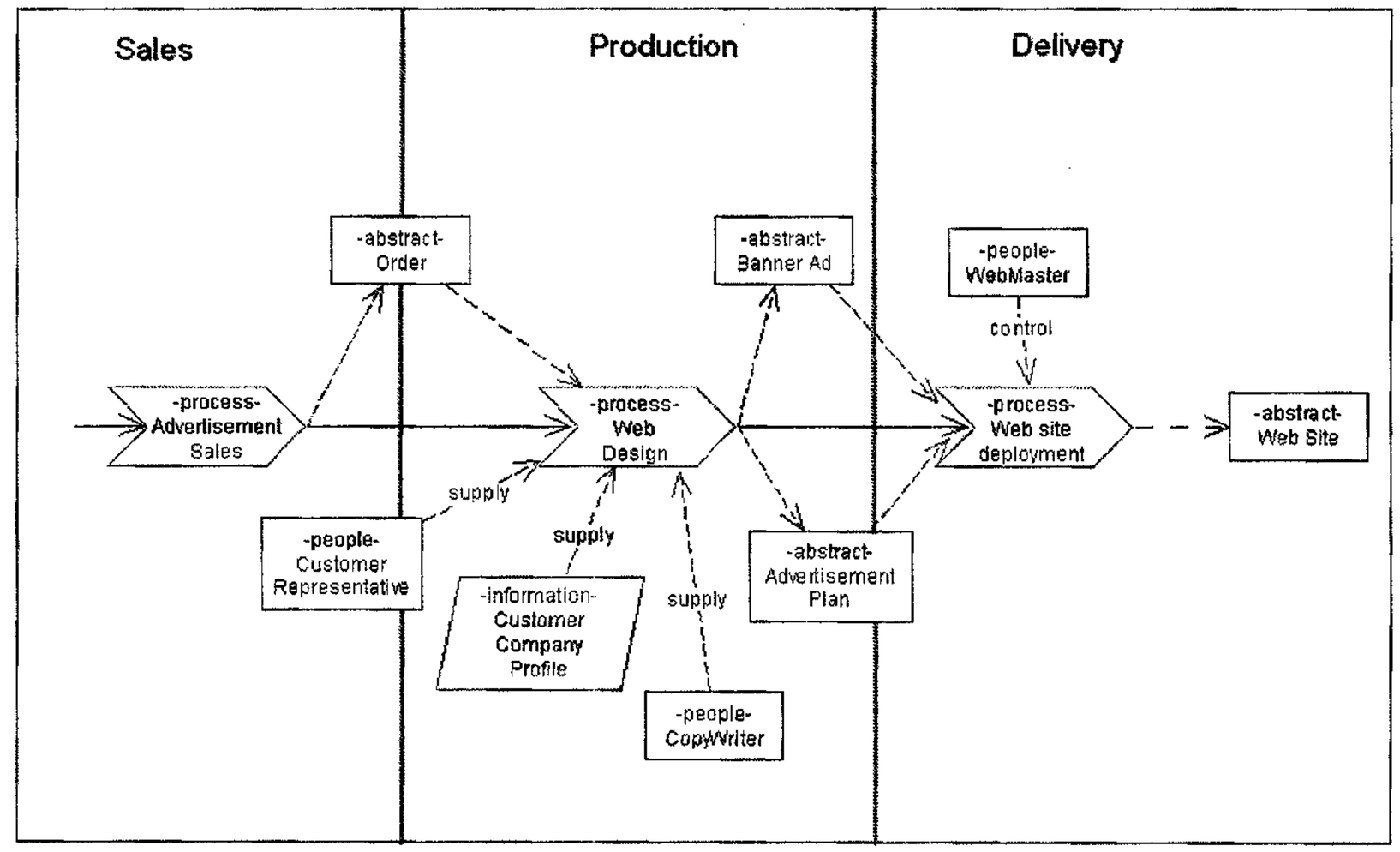
In UML, a business process is represented in an activity diagram. A process is an activity stereotyped to process. The following figure shows a generic process diagram:



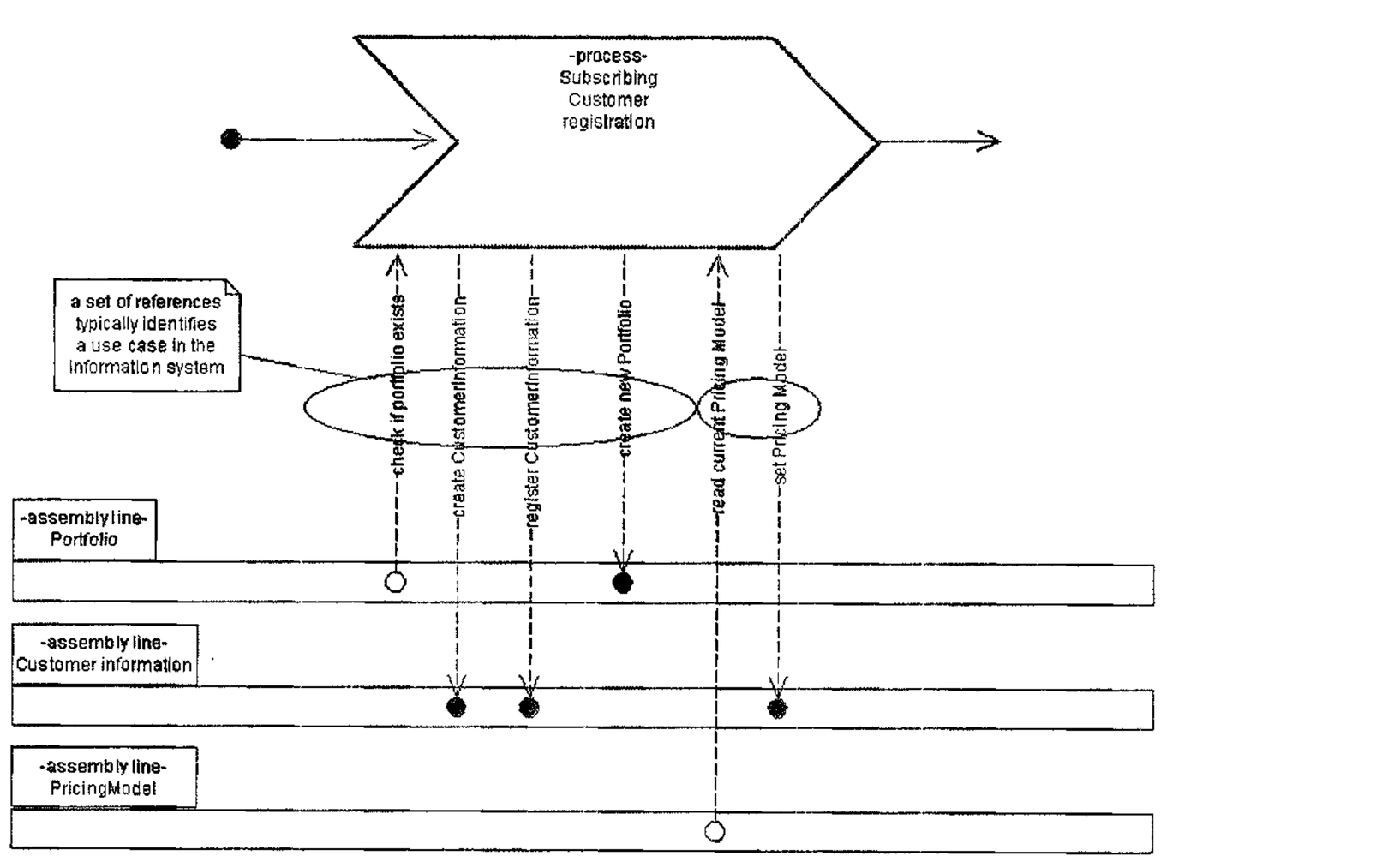
Resource objects and goal objects that are involved in the process are placed around the process. These objects are:

- Goal objects: as previously defined.
- Input objects: objects that are either consumed or refined in the process.
- Output objects: objects that are produced by the process.
- Supplying objects: resources that are participating in the process but are not refined or consumed.
- Controlling objects: resources that control or run the process.

The next figure shows a simple example of a business process:



A variation of the activity diagram used by Eriksson-Penker method is the Assembly Line Diagram in which the assembly lines are used to represent objects in an information system. The next figure illustrates this:



In this diagram, we see that a set of references become a use case that the information system has to provide. This is very important because it maps the business process to use cases that describe the functional requirements of an information system. It can also help identify the proper actors if the use case – the roles played by the process that uses the assembly lines. Assembly line

diagrams provide the connection between business modeling and software system requirements modeling with use cases. A common question when modeling use cases in a software system is: how do I know I have defined the right use cases in terms of the business? The assembly line diagram is a good technique to answer this.

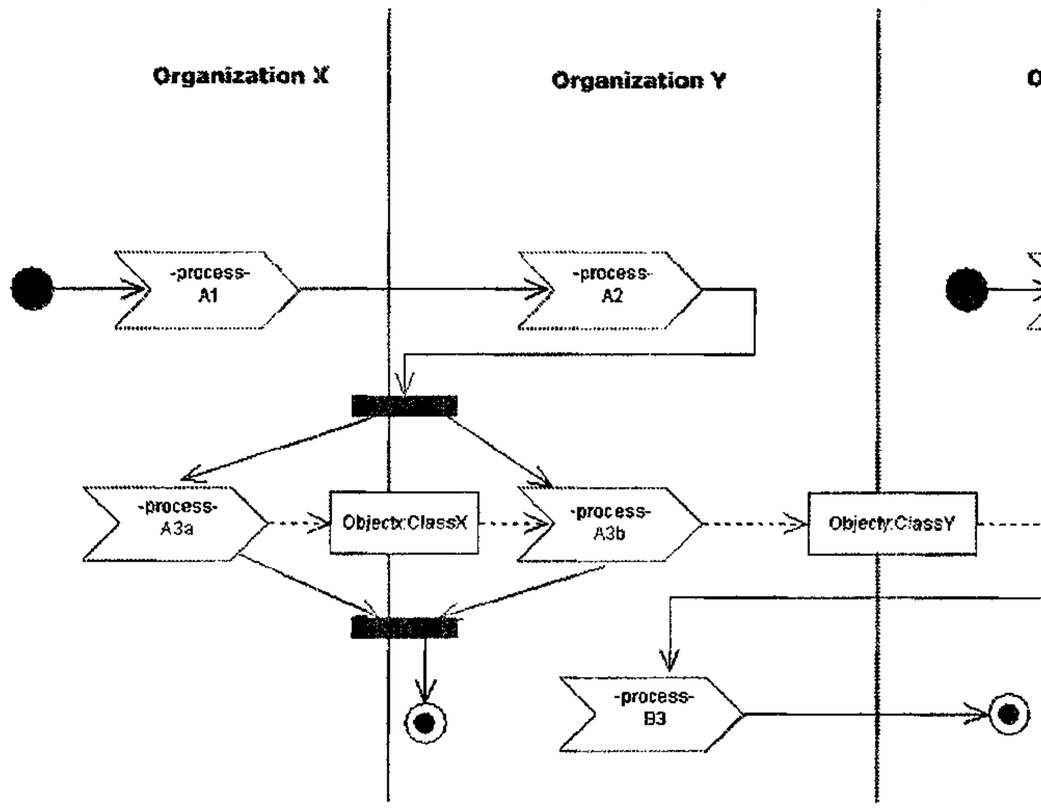
By deriving all requirements of the support systems from the processes, with the assembly line diagrams and the process properties, it is also possible to verify them. Moreover, the processes themselves result from goal modeling, which means that the system requirements can be validated against the overall business goals! By working with goals, processes, and the assembly line diagram, it is possible to verify and validate both functional and nonfunctional system requirements.

A.4.3 Organizational view

The Organizational/Structure view shows the structures of the resources, the products, or the services and the information in the business, including the traditional organization of the company (divisions, departments, sections, etc.). This view is considered supplemental to the Process view, depicting information that can't be shown in the process diagram but that is vital to the operation of the company. The UML diagrams used to document this view are class and object diagrams. These diagrams are very important and should allow the identification of the static structure of an information system.

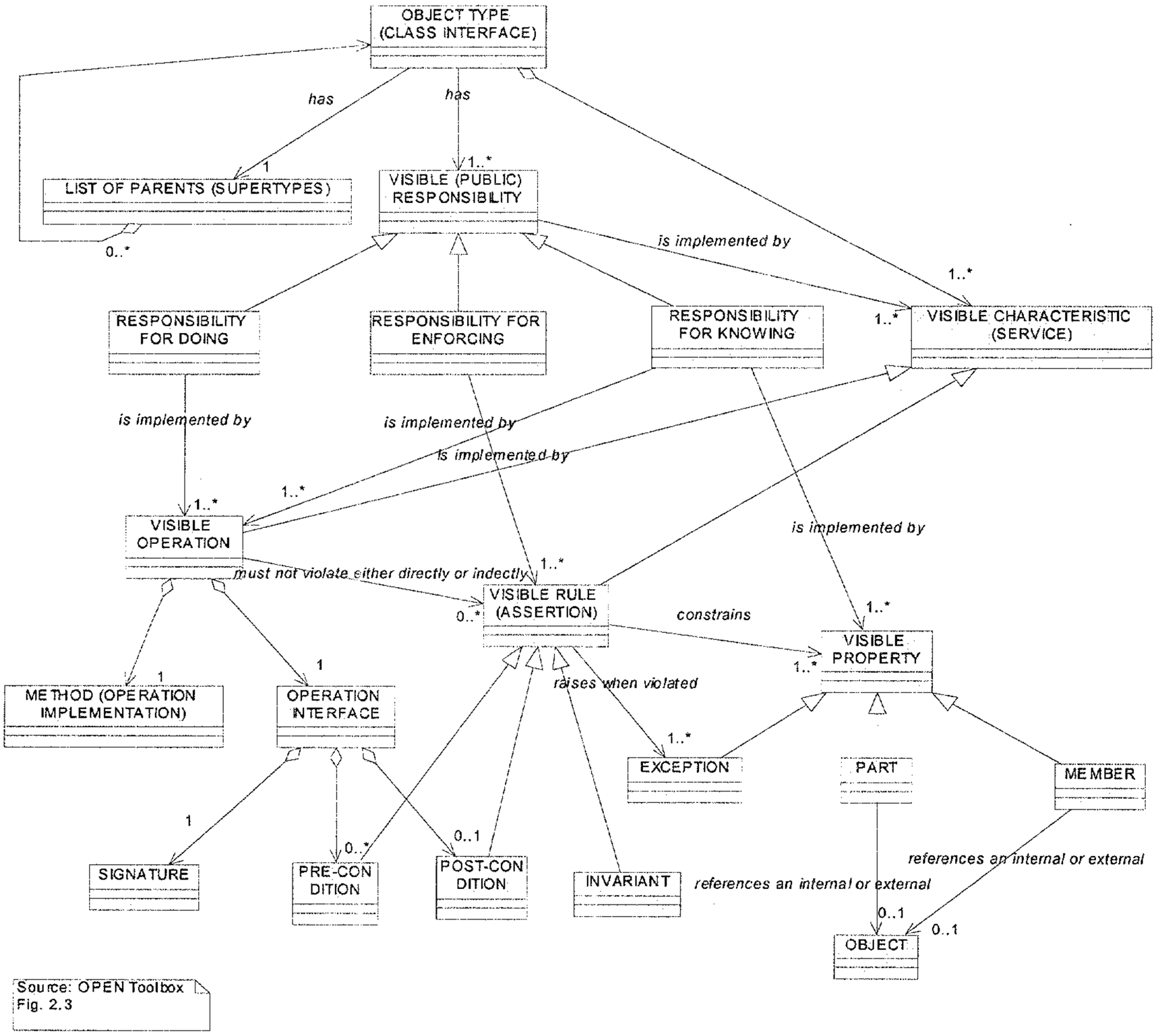
A.4.4 Behavior view

The Behavior view illustrates both the individual behavior of resources in the business as well as the interaction between several different resources and processes. The behavior of the resource objects is governed by the Process view, which shows the overall main control flow of the work performed. The Business Behavior view looks into each of the involved objects in more detail: their state, their behavior in each state, and possible state transitions. The Behavior view also shows the interaction among different processes, such as how they are synchronized. The UML diagrams used in this view are the Statechart diagram, the Sequence and Collaboration diagrams, and the Process diagrams. An example of this is shown in the following figure:



A.5 From a business model to a system model

The following diagrams illustrates the conversion from a business object that has responsibilities to a UML class with operations, attributes and rules:



Appendix B: Model Driven Architecture

B.1 MDA Related Standards

MDA is one of the emerging standards controlled by the Object Management Group (OMG), an open membership, not-for-profit consortium that produces and maintains computer industry specifications for interoperable enterprise applications. Some other MDA related standards include:

Standard	Description
UML (OMG)	Unified Modeling Language
MOF (OMG)	Meta Object Facility
JMI (Sun)	Java Metadata Interface
XMI (OMG)	XML Model Interchange
QVT (OMG)	Query, Views and Transformations Standard.
OCL (OMG)	Object Constraint Language
CWM (OMG)	Common Warehouse Metamodel

B.1.1 UML

OMG's UML is the key enabling technology for the Model Driven Architecture: every application using the MDA is based on a normative, platform-independent UML model. UML has been defined as a MOF metamodel.

B.1.2 MOF

The MOF is the OMG's Meta-Object Facility standard. It is a language for defining other languages (meta-information). For example, programming language types, database schemas, UML models, etc

The OMG defines a 4 level metamodel architecture:

Meta Level	MOF TERMS	Examples
M3	Meta-Metamodel	MOF Class
M2	Meta-Metadata or Metamodel	UML Class
M1	Metadata or Model	UML BankAccount Class
M0	Data	Joe Smith's BankAccount

Given the well-defined and limited goal of MOF, the OMG selected a small subset of UML to define MOF and uses standard UML notation.

MOF is a foundation technology for MDA because it offers a common way of describing models. MOF makes it easier to develop and specify the necessary rules to map the abstract models all the way to the implementation models.

B.1.3 JMI

The Java Metadata Interface is a native Java interface for retrieving information about M1 models from a MOF repository. It is described in the Sun Microsystems' Java Metadata Interface Specification.

B.1.4 XMI

XMI defines an XML-based interchange format for UML metamodels and models (since a metamodel is just a special case of a model), by standardizing XML document formats and DTDs. In so doing, it also defines a mapping from UML to XML.

XMI's goal is to facilitate the exchange of model information between different MDA tools and technologies.

B.1.5 QVT

This standard addresses the way transformations are achieved between models whose languages are defined using the MOF. It consists of:

- A language for creating views on model

- A language for querying model
- A language for writing transformation definitions

B.1.6 OCL

The Object Constraint Language (OCL) is part of the UML and can be used to specify all kinds of constraints, pre- and post-conditions, guards etc. over the objects in the different models.

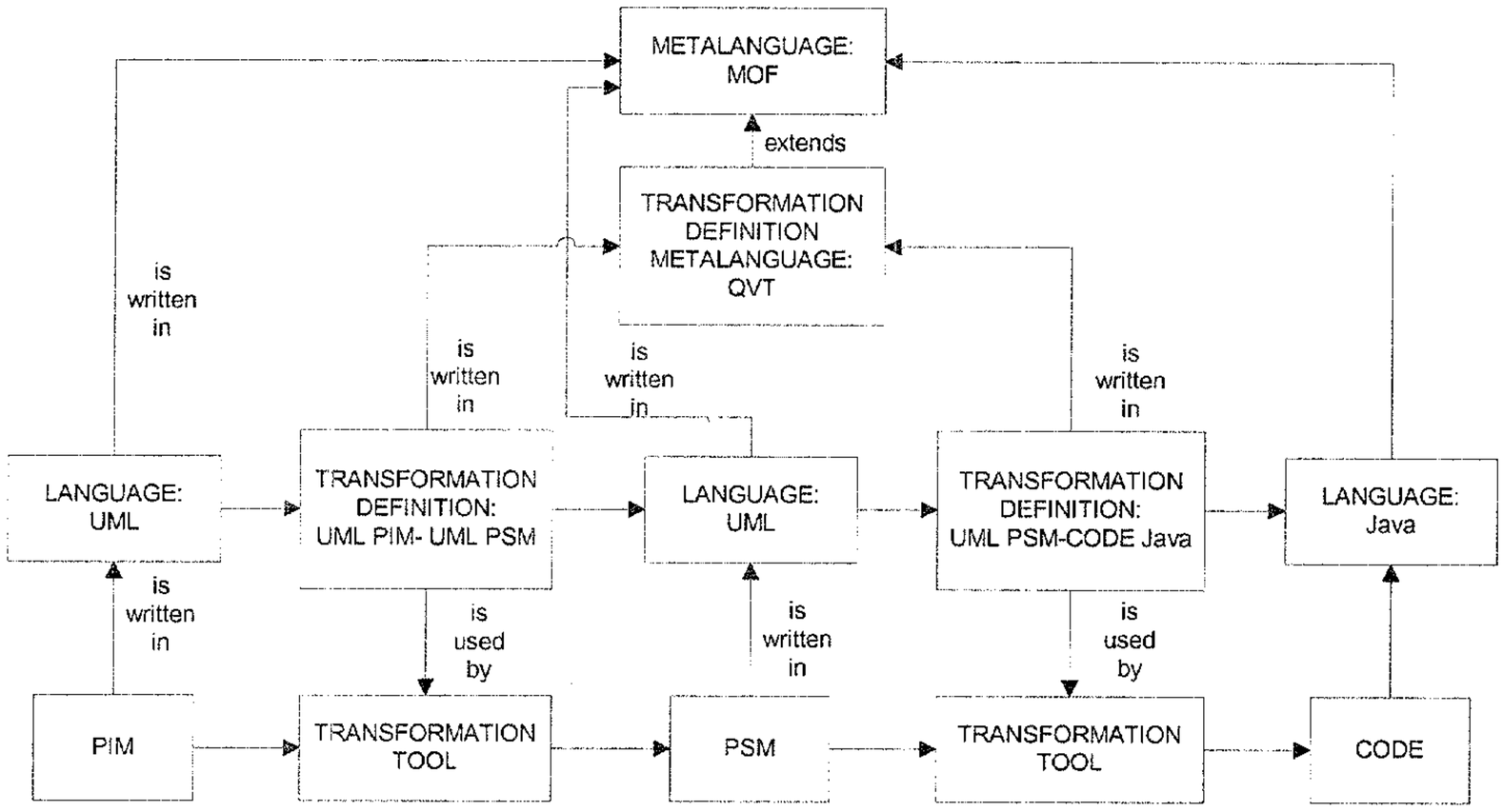
B.1.7 CWM

CWM is a language or framework for specifying the external presentation of metadata for purposes of interchange. CWM is implemented as a MOF metamodel (M2). CWM is used essentially for constructing Data Warehouse & Business Intelligence models.

It provides

- A standard language for defining structure and semantics of metadata in a formal way (UML/OCL)
- A standard interchange mechanism for sharing metadata defined in the standard language (XML/XMI)
- A standard specification (interface) for access to, and discovery of, the metadata defined in the standard language (IDL)

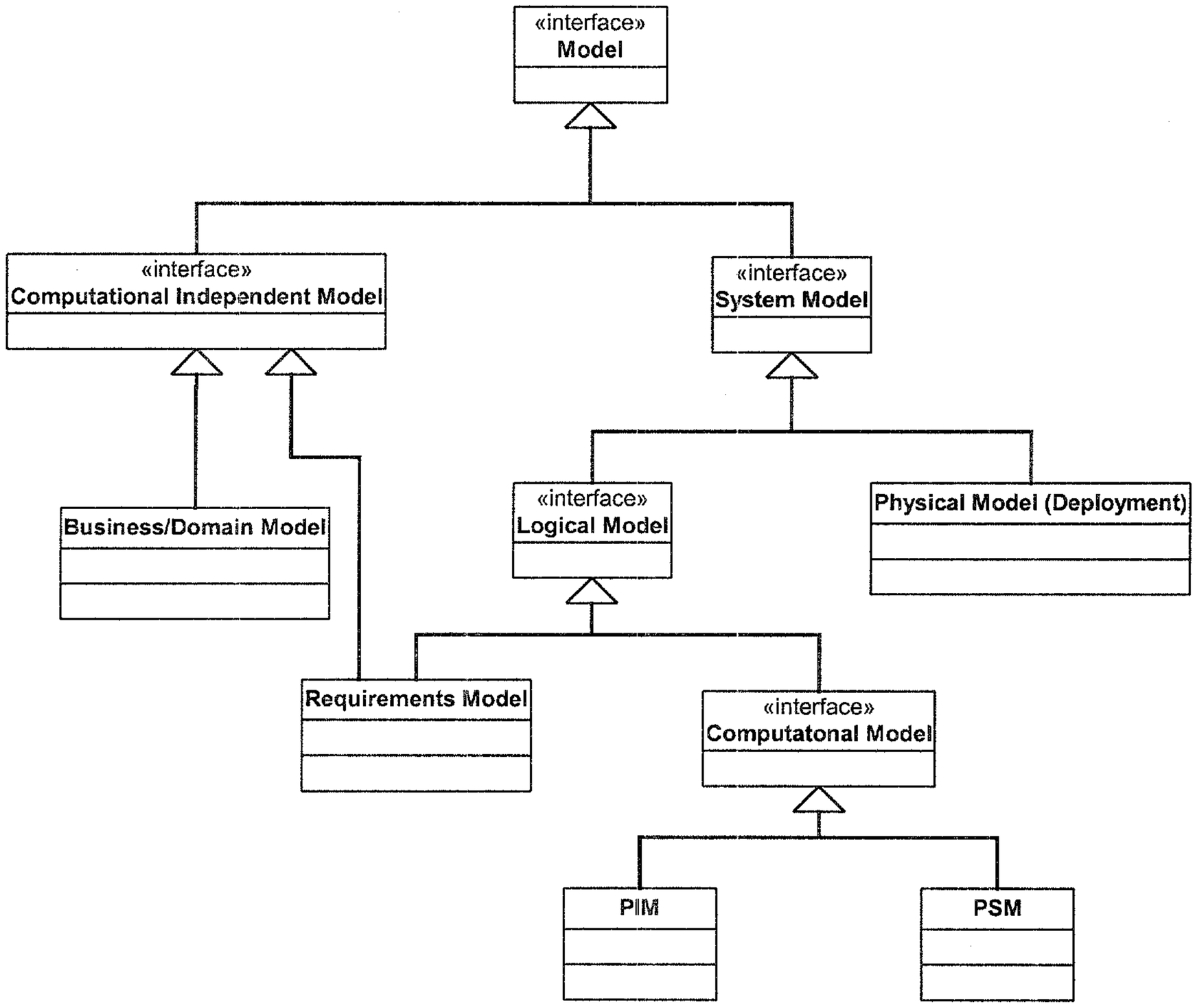
B.2 MDA Framework



The preceding figure presents a picture of the complete MDA framework. It shows how a Process Independent Model is transformed using a transformation tool into a Process Specific Model and then into actual code. The framework introduces the idea of transformation definitions that are used by the transformation tool during the transformation process. The transformation definitions are themselves defined using a transformation definition language that is an extension of the MOF meta-language.

B.3 MDA Models

B.3.1 Model Definitions



The preceding figure depicts a taxonomy of different possible kinds of models used in software development.

- **Business/Domain models** are Computational Independent Models (CIM) and describe aspects of the business, irrespective of where those aspects are slated to be automated.
- **System models** describe aspects of a computer system that automates elements of the business.
- **Logical models** describes the logic of a system via behavioral and static models

- **Physical models** describes physical artifacts used during development and runtime
- **Requirements models** are also Computation Independent Models. They describe the logical system rather than the business but in a computation independent fashion
- **Computational models** describe the logical system as well but take technical factors into account. These models provide the information necessary for generators
- **A Computational Independent Model (CIM)** is a software independent model used to describe a business system. A CIM may be supported by software, but the CIM itself remains independent of software
- **A Platform Independent Model (PIM)** is also a computational model but is independent of the platform specific information of its derived PSM. Platform independence is a relative term. A model can be independent of the OS, the implementation language, the middleware... Hence, PIMs can be found at multiple levels and can take into account some technical factors.
- **A Platform Specific Model (PSM)** is a computational model depends on the specification of a reference set of platform specific technologies. PSMs are derived from PIMs and take additional technical factors specific to the platform into consideration. In MDA, code is considered to be a Platform Specific Model.

B.4 MDA Limitations

B.4.1 Business Modeling

MDA does not explicitly address the complete development lifecycle of business systems since it does not provide a formal model for business and requirements modeling. We believe this is a substantial limitation because of the iterative nature of software development where the feedback loop actually involves interaction all the way up to the business model, not just up to analysis.

Although there is a move in UML 2.0 to start supporting such models, MDA tools are just beginning to incorporate business and requirements models into the development lifecycle.

B.4.2 Complex Formal Models

Although the guiding principles behind the MDA are very simple, implementing MDA is currently a very difficult proposition. The models at the core of the abstraction mechanism of MDA must be formal and complete in order to allow tools to be able to generate complete applications. While the models do raise the level of abstraction, the level of formalism MDA requires can potentially offset the advantages.

If this complexity is not addressed and shielded from the end user, this could delay substantially and potentially kill the adoption of the technology, causing MDA to suffer the same fate as the early CASE tools of a few years ago. We believe a good MDA tool should provide extensive facilities to shield users from the complexity of creating those formal models.

B.4.3 Open-ended Tools

Existing UML modeling tools on the markets are ill conceived to support the rigor necessary to produce the formal models required by MDA. It is possible with those tools to create models that contain inconsistencies or do not completely define the relationships of the system entities. This was fine when the models where used informally, but simply cannot work in the MDA context. MDA tools need the formality in order to be able to automate the generation of executable code.

B.4.4 Evolving Standards

We are still in the infancy of MDA, standards are still evolving and being challenged by different alternatives. It will be necessary for MDA vendors to shield the community from changes in those standards as much as possible.

B.4.5 Incomplete Support of Dynamic Models

The weak area in MDA is in the behavioral or dynamic part. UML includes many different diagrams to model dynamics, but their definition is not formal and complete enough to enable the generation of a PSM. Some work has been done on Executable UML using state machines, but this is mostly the realm of embedded software development.

Another initiative, the Action Semantics (AS) defines a fairly low level language that is situated at the same level of a PSM. Therefore, using Executable UML has little advantage over writing the dynamics of the system in the PSM itself. Furthermore, AS has not been standardized by the OMG.

B.4.6 User Interface Modeling

Other than at an architectural level, MDA does not address the actual development of a system's user interface. Some MDA tools currently generate some generic user interfaces based on some well-known UI patterns, but they produce unsophisticated interfaces, which are largely insufficient in most cases. We believe the integration of a user interface modeling and layout tool is absolutely essential in providing a viable MDA solution that spans the entire lifecycle. Failure to do so will severely limit the acceptance of MDA for domains outside the embedded systems world.

GLOSSARY

ARAD: Architected Rapid Application

BM: Business Modeling

BPM: Business Process Management

CWM: Common Warehouse Metamodel

JMI: Java Metadata Interface

MDA: Model Driven Architecture

MOF: Meta Object Facility

PIM: Platform Independent Model

PSM: Platform Specific Model

QVT: Query, Views and Transformations Standard.

RAD: Rapid Application Development

RUP: Rational Unified Process

UML: Unified Modeling Language

XMI: XML Model Interchange

REFERENCES

[DARN 1997] G. Darnton, *Business Process Analysis,* Addison-Wesley, 1997.

[DAVEN 1992] T.Davenport, *Process Innovation: Reengineering Work through Information Technology,* Cambridge, 1992 .

[DAVIS 2001] R.Davis, *Business Process Modeling with ARIS,* Springer, 2001.

[ERIK 2000] H.E. Eriksson and M. Penker, *Business Modeling with UML: Business Patterns at Work,* John Wiley & Sons, 2000.

[FOW 2002] M. Fowler, *Analysis Patterns, reusable object models,* Addison-Wesley, 2002.

[FRANK 2003] D. S. Frankel, *Model Driven Architecture: Applying MDA to Enterprise Computing,* John Wiley & Sons, 2003.

[GAMMA 1995] Gamma, Erich, Helm, Vlissides, Johnson, *Design Patterns,* Addison-Wesley, 1995.

[GRAHAM 1997] I. Graham, *The OPEN Process Specification,* Addison-Wesley, 1997.

[HUB 2002] R. Hubert, *Convergent Architecture: Building Model-Driven J2EE Systems with UML,* John Wiley & Sons, 2002.

[JENZ 2003] D. Jenz, *Simplifying the Software Development Value Chain Through Ontology-Driven Software Artifact Generation,* 2003.

[MELLOR 2002] S. Mellor and M. Balcer, *Executable UML: A foundation for Model-Driven Architectures,* Addison-Wesley, 2002.

[STEVENS 2003] M. Stevens, *Service-Oriented Architecture Introduction,* Developer.com, 2003.

[TAYLO 1995] D. A. Taylor, *Business Engineering with Object Technology,* John Wiley & Sons, 1995.