MDA

Model Driven Architecture

Joaquin Miller
Lovelace Computing
representing X-Change Technologies

This is a tutorial prepared for the OMG Fourth Workshop
UML™ for Enterprise Applications: Delivering the Promise of MDA

The tutorial describes the MDA pattern and the basic concepts on which it is built. It discusses MDA platform-independent to platform-specific transformation and briefly outline other MDA capabilities.

The material for this presentation is based on the current MDA Guide, omg/2003-06-01. http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf

http://www.joaquin.net/ODP/

Joaquin Miller joaquin.no.spam that-sign acm the-dot org
Lovelace Computing Company www.joaquin.net

Acknowledgements

This is largely a presentation of the OMG MDA Guide.

The Guide was prepared by the ORMSC, under the supervision of the AB.

Many folk contributed to the Guide.

I'm to blame for what is presented here that is not in the Guide (and for what is in the Guide).

Lovelace Computing

Mariano Belaunde (France Telecom R&D)
Cory Casanave (Data Access Technologies)
Desmond DSouza (Kinetium)
William El Kaim (BusinessOne/Thales)
William Frank (X-Change Technologies)
Randall Hauch (Metamatrix)
Matthew Hettinger (Mathet Consulting)
Duane Hybertson (MITRE)
Jean Jourdan (THALES)
Toshiaki Kurokawa (CSK Corp.)
Stephen Mellor (Project Technology)
Jeff Mischkinsky (Oracle)
Chalon Mullins (Charles Schwab)
Laurent Rioux (THALES)
Ed Seidewitz (Intelidata Technologies Corporation)
Jon Siegel (OMG)
Dave Smith (Deere & Company)
Akira Tanaka (Hitachi)
Axel Uhl (Interactive Objects Software)
Dirk Weiseand (Interactive Objects Software)
Carol Burt (2AB)
Fred Cummins (EDS)
Keith Duddy (DSTC)
Alan Kennedy (Kennedy Carter)
David Frankel (David Frankel Consulting)
Stan Hendryx (Hendryx & Associates)
Richard Hubert (Interactive Objects Software)
Sridhar Iyengar (IBM)
Thomas Koch (Interactive Objects Software)
Anthony Mallia (CIBER)
Joaquin Miller (Lovelace Computing)
Jishnu Mukerji (HP)
Makoto Oya (Hitachi)
Peter Rivett (Adaptive)
Bran Selic (Rational Software)
Oliver Sims (Sims Associates/IONA)
Richard Soley (OMG)
Sandy Tyndale-Biscoe (OpenIT)
Andrew Watson (OMG)
Bryan Wood (OpenIT)

Communication

= Question or objection

= Uncertainty

= Agreement or acceptance

Lovelace Computing

Communication, please!

= Question or objection

= Uncertainty

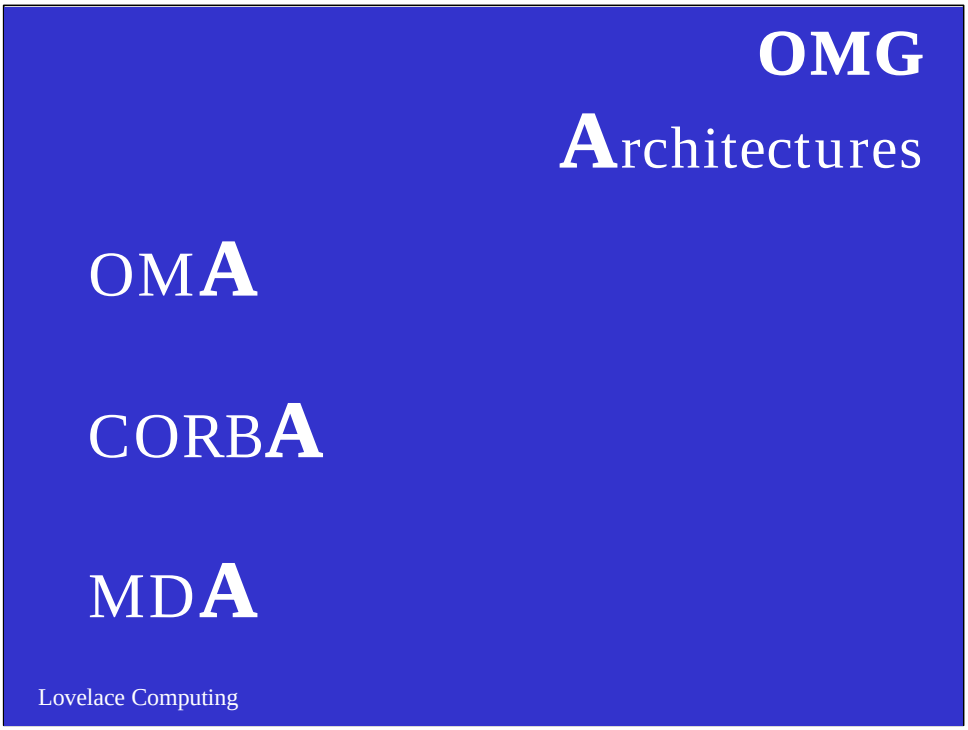
= Agreement or acceptance

Lovelace Computing

Communication, please!

= contribution

Lovelace Computing

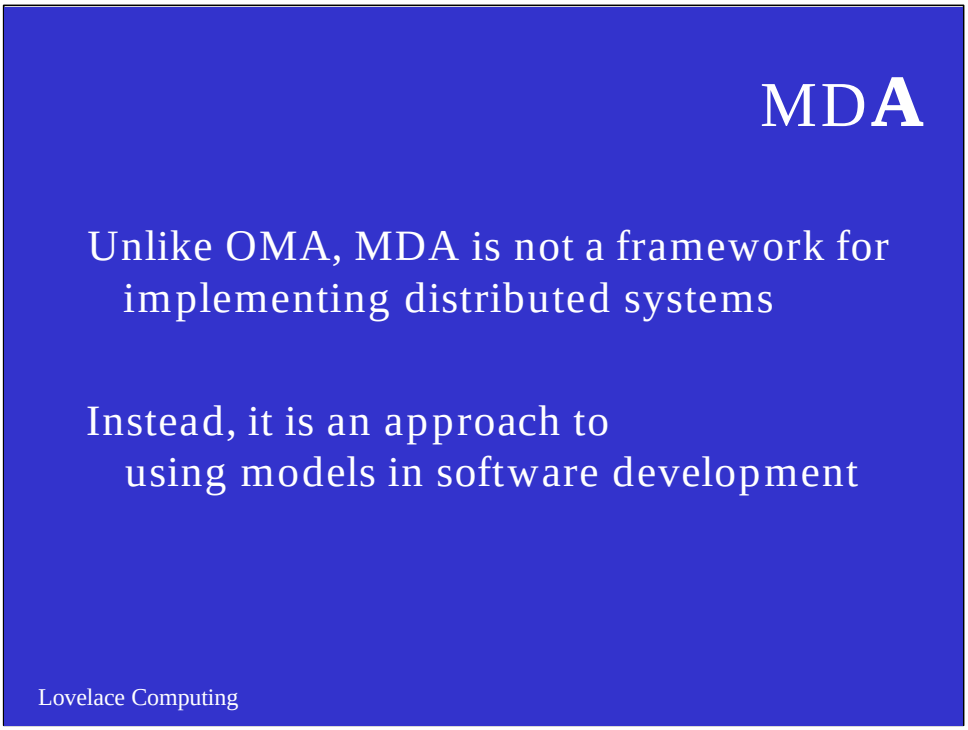


Over the last dozen years, the Object Management Group, better known as OMG, standardized the object request broker (ORB) and a suite of object services. This work was guided by the Object Management Architecture (OMA), which provides a framework for distributed systems and by the Common ORB Architecture, or CORBA™, a part of that framework.

The OMA and CORBA were specified as a software framework, to guide the development of technologies for OMG adoption. This framework is in the same spirit as the OSI Reference Model and the Reference Model of Open Distributed Processing (RM-ODP or ODP). The OMA framework identifies types of parts that are combined to make up a distributed system and, together with CORBA, specifies types of connectors and the rules for their use.

Six years ago Mary Loomis led the OMG members in further enlarging their vision to include object modeling. This resulted in the adoption of the Unified Modeling Language, UML. OMG members then began to use UML in the specification of technologies for OMG adoption.

In keeping with its expanding focus, last year OMG adopted a second framework, the Model-Driven Architecture™ or MDA™ [6]. Development of technologies for this framework is ongoing.



MDA is not, like the OMA and CORBA, a framework for implementing distributed systems. It is an approach to using models in software development.

MDA is another small step on the long road to turning our craft into an engineering discipline.

Three primary goals of MDA are

— portability,
— interoperability and
— reusability

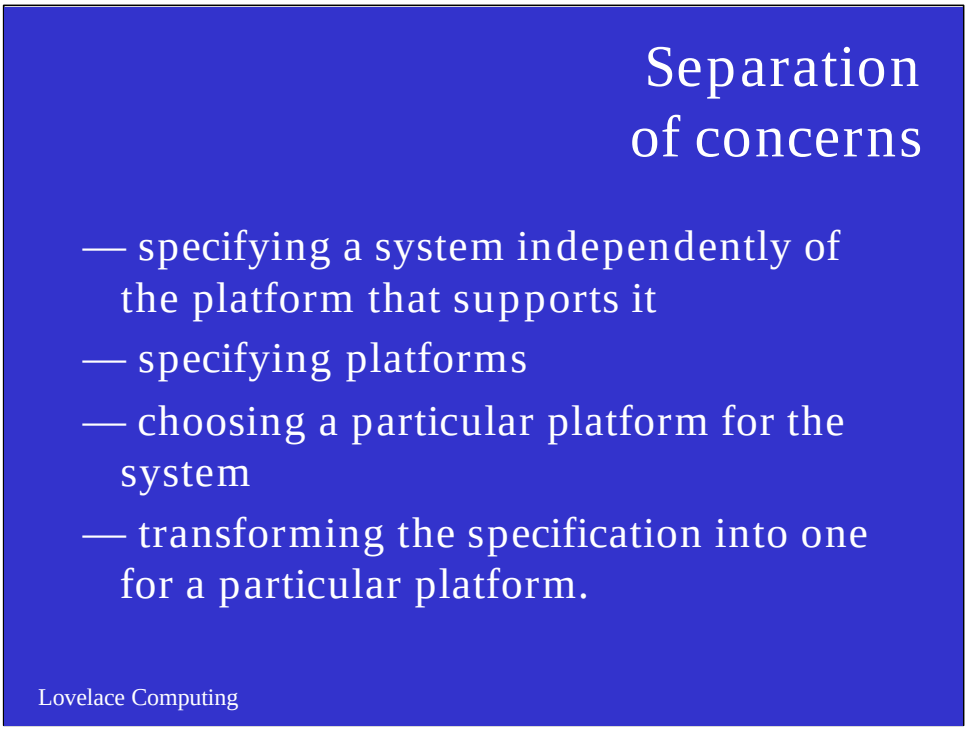
Two other goals are

— increased quality

— reduced cost

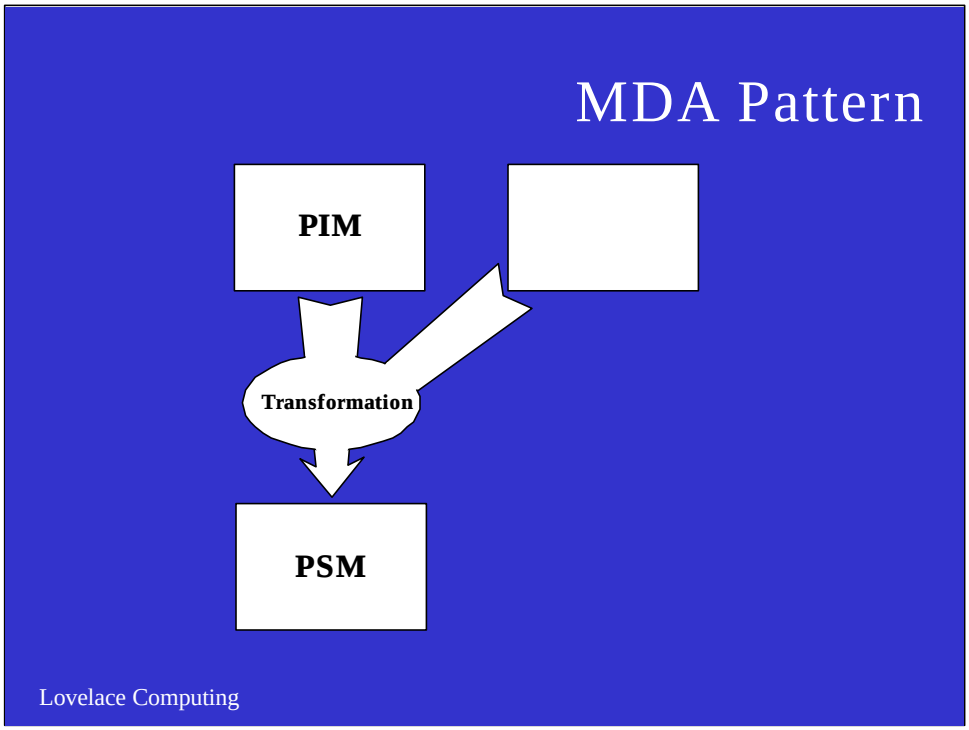
Despite the wrong thinking of many panic stricken project managers, these two goals go hand in hand. Increased quality results in lower cost. Project cost is reduced. And, even more important for the owner, total cost of ownership is reduced dramatically.

MDA works toward the achievement of these goals through architectural separation of concerns.



The Model-Driven Architecture starts with the well-known and long established idea of separating the specification of the operation of a system from the details of the way that system uses the capabilities of its platform.

MDA provides an approach and tools for:
— specifying a system independently of the platform that supports it,
— specifying platforms,
— choosing a particular platform for the system, and
— transforming the specification into one for a particular platform.



Model transformation is the process of converting one model to another model of the same system.

The drawing illustrates the MDA pattern, by which a PIM is transformed to a PSM.

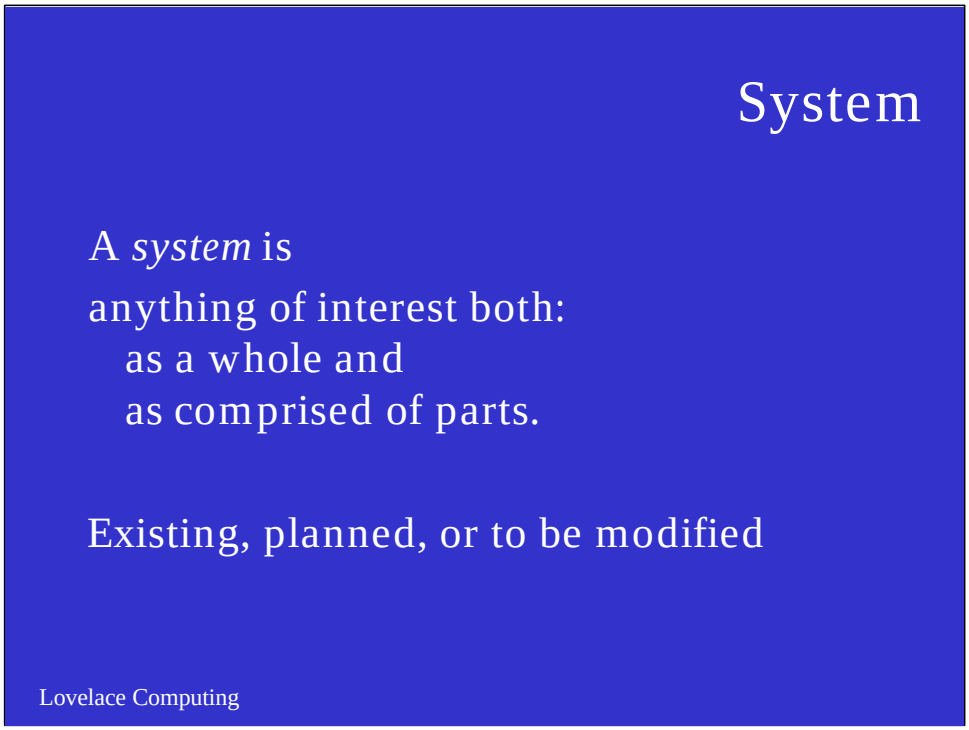
The drawing is intended to be suggestive. The platform independent model and other information are combined by the transformation to produce a platform specific model.

The drawing is also intended to be generic. There are many ways in which such a transformation may be done. However it is done, it produces, from a platform independent model, a model specific to a particular platform.

The box with no name represents what goes into the transformation, in addition to the platform independent model. This varies with different styles of MDA.

Basic Concepts

- System and model
- Model-driven
- Architecture
- Viewpoint
- Platform
- Independence
- Transformation

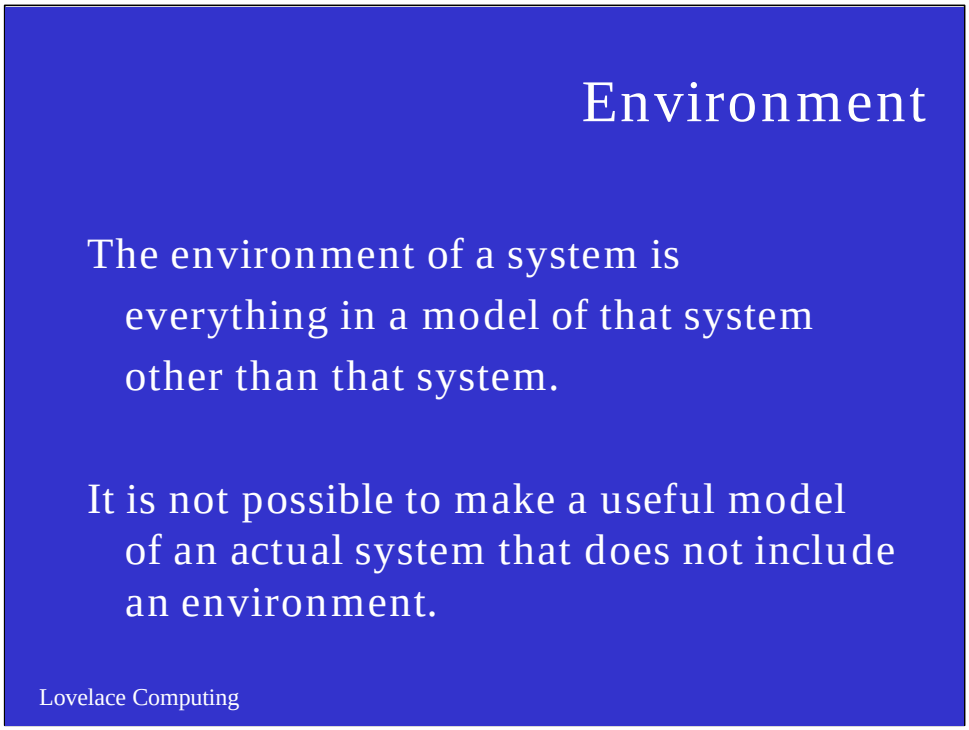


We'll present the MDA ideas in terms of some existing or planned system. That *system* may include anything: a program, a single computer system, some combination of parts of different computer systems, a federation of computer systems, each under separate control, people, an enterprise, a federation of enterprises…

System: Something of interest as a whole or as comprised of parts. A component of a system may itself be a system, in which case it may be called a subsystem.

[RM-ODP 2-6.5 www.joaquin.net/ODP/Part2/6.html#6.5]

The central focus of MDA is software. Much of the discussion will focus on software within automatic information processing systems.



UML will be used to specify or describe open systems: those that interact with their environment.

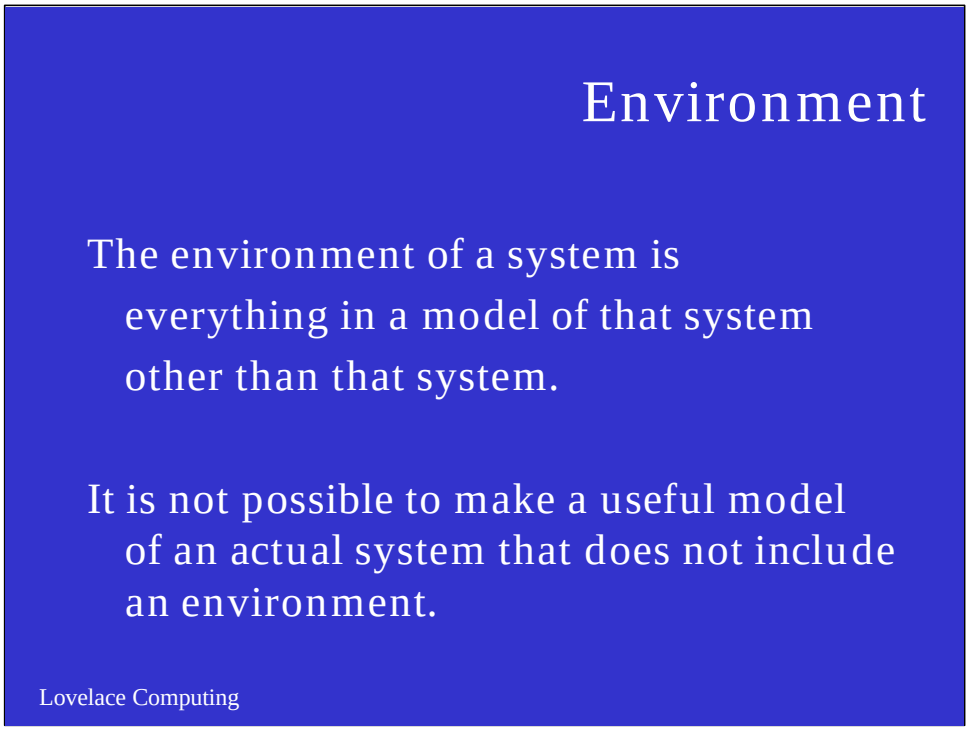
So, in order to understand an existing system, parts of the environment of that system must be described. And parts of the environment of a system to be built must be specified, in order to understand what the environment must be like for the system to work.

In an ODP or CommunityUML model, the environment of a system is everything in the model, other than that system.

Environment (of an object): the part of the model which is not part of that object.

[RM-ODP 2-8.2 www.joaquin.net/ODP/Part2/8.html#8.2]

Of course, the system and its environment form another system. When the distinction does not matter, it is not this larger system we mean in this presentation, but the system being specified or described.



The following statement is just fine in practice, though not exactly true:

It is not possible to make a useful model of an actual system that does not include an environment.

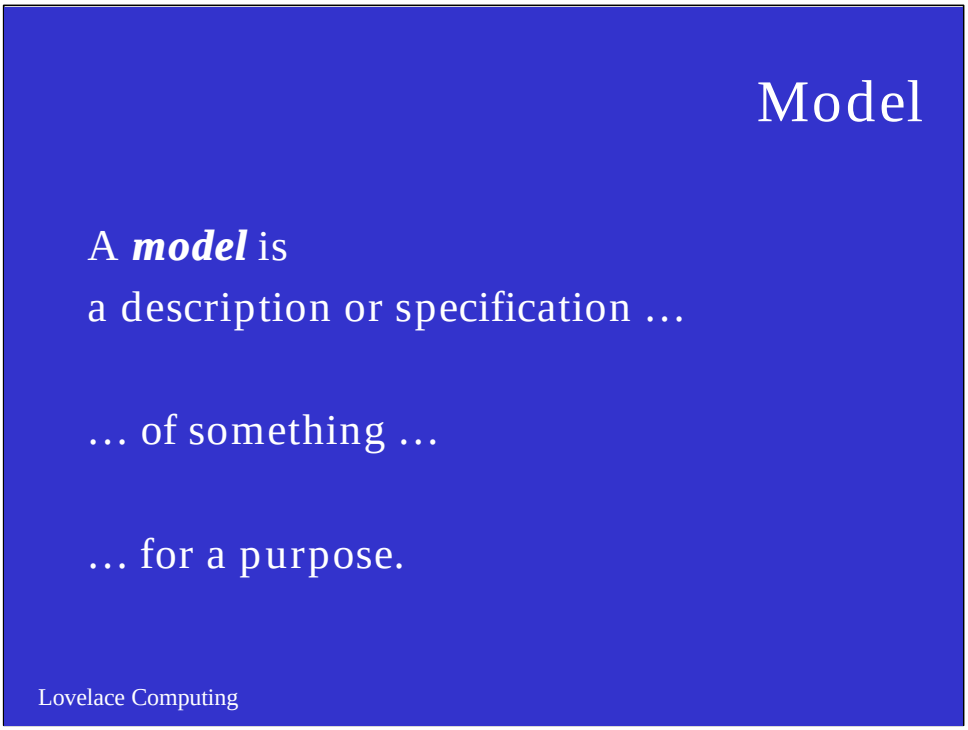
The reason that the statement is not exactly true is that the universe is a closed system, so has no environment.

The universe the only system that is closed at all times. Other than the universe, every system is open. [19]

[19] Bunge, Treatise on Basic Philosophy ISBN 90-277-0944-0 (volume 4)

In this presentation we will consider that a system consists of one or more ***applications***, supported by one or more platforms

'Application' is not intended as a technical term, but we will need it later in the discussion.



A *model* of a system is a description or specification of that system and its environment for some certain purpose. A model is a model of something.

A model is often presented as a combination of drawings and text. The text may be in a modeling language or in a natural language.

The more exact the model, the more likely the system built using the model will be what is wanted.

Model

A ***model*** is
a description or specification …

… **of** something …

… for a purpose.

Lovelace Computing

Model

A ***model*** is
a description or specification …

… **of something** …

… for a purpose.

Lovelace Computing

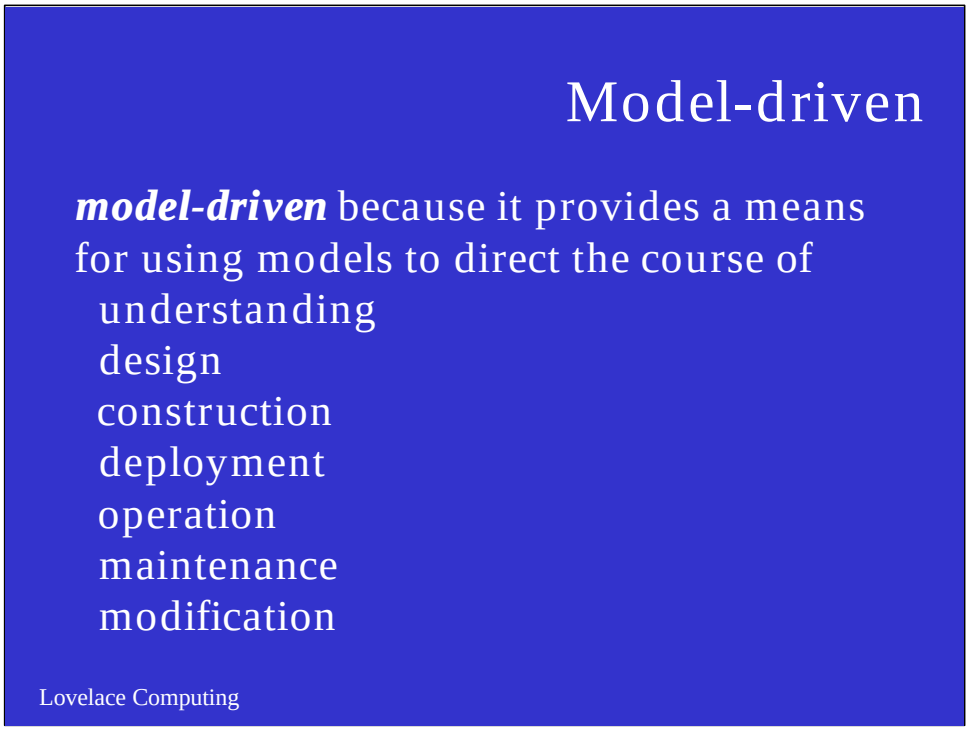
Model

A model is
a description or specification …

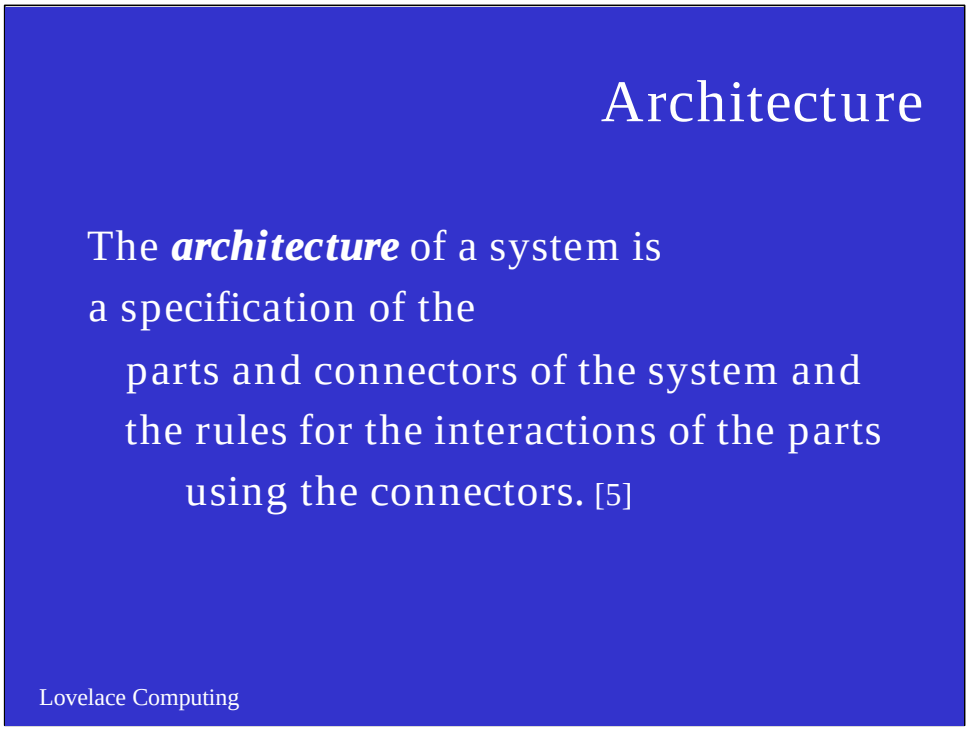
… of something …

… for a purpose.

Lovelace Computing



MDA is an approach to system development, which increases the power of models in that work. It is *model-driven* because it provides a means for using models to direct the course of understanding, design, construction, deployment, operation, maintenance and modification.



The *architecture* of a system is a specification of the parts and connectors of the system and the rules for the interactions of the parts using the connectors. [5]

Architecture (of a system): A set of rules to define the structure of a system and the interrelationships between its parts.

[RM-ODP 2-6.6 www.joaquin.net/ODP/Part2/6.html#6.6]

The Model-Driven Architecture prescribes certain kinds of models to be used, how those models may be prepared and the relationships of the different kinds of models.

The 'architecture' in 'model driven architecture' extends the central meaning of architecture.[18] It is about the architecture of a system of models. The models are connected by transformation relationships, which structure the models.

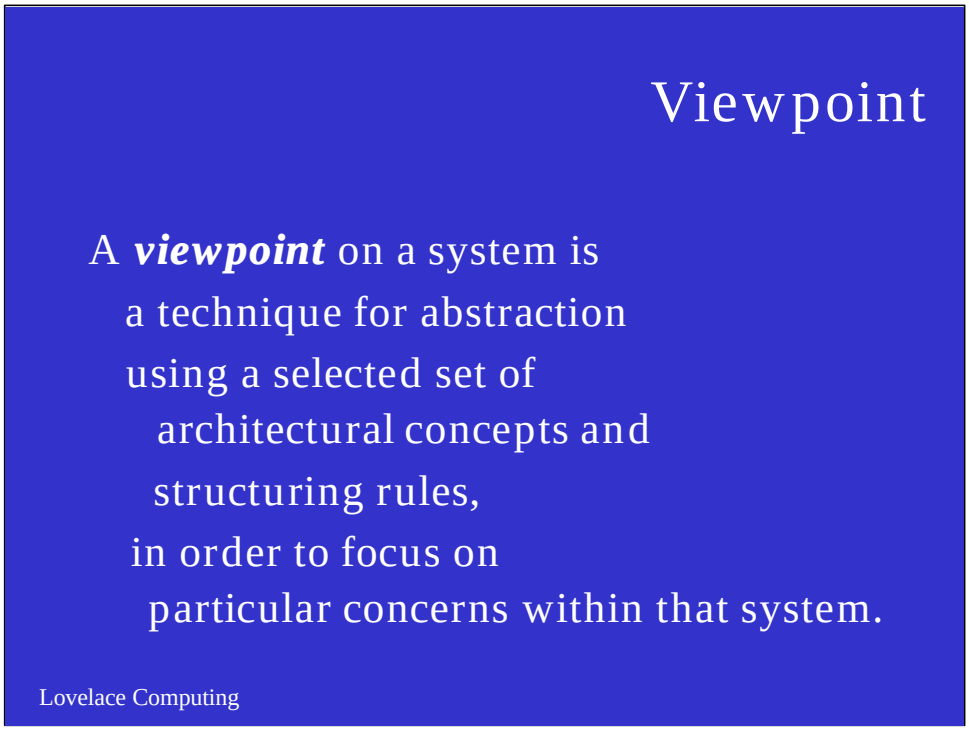
[5] Shaw and Garlan, Software Architecture, Prentice Hall ISBN 0-13-182957-2

[18] Lackoff, Women, Fire, and Dangerous Things, University of Chicago, ISBN 0-226-46804-6

Architecture

The ***architecture*** of a system is
a set of rules to define
the structure of a system and the
interrelationships between its parts. [1]

Lovelace Computing



A *viewpoint* on a system is a technique for abstraction using a selected set of architectural concepts and structuring rules, in order to focus on particular concerns within that system. [Here 'abstraction' is used to mean the process of suppressing selected detail to establish a simplified model.]

The concepts and rules may be considered to form a viewpoint language.

Examples:

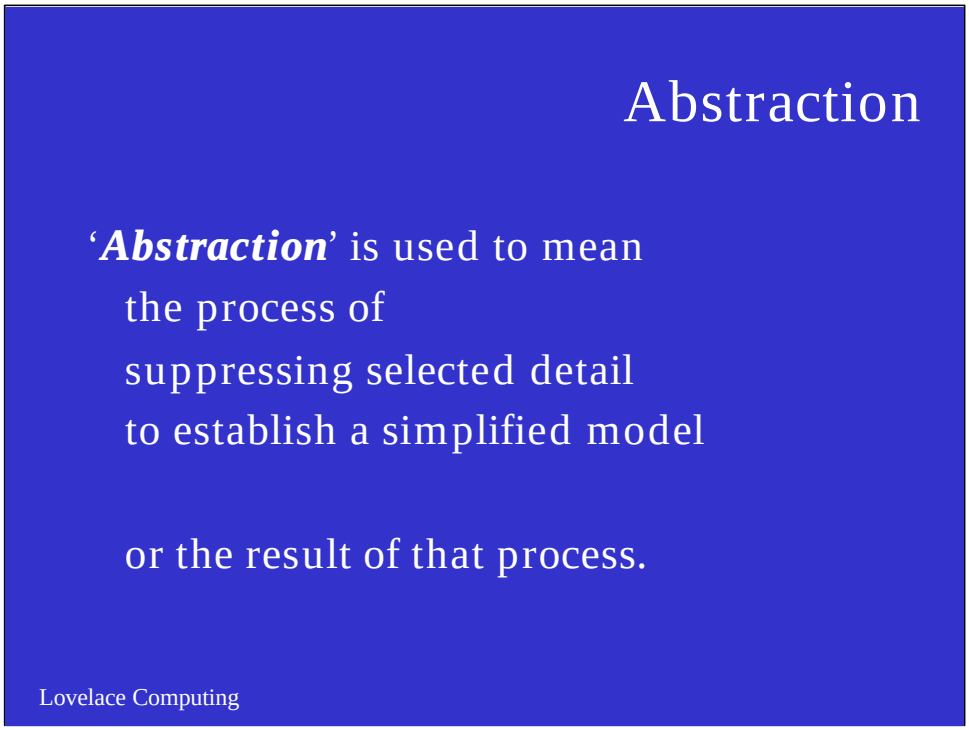
The Reference Model of Open Distributed Processing (ODP) provides five viewpoints for specifying a distributed system. [1]

Another classification specifies three (very similar to the SPARC database model viewpoints [2]): a conceptual viewpoint, describing the place of a system in the situation in which that system will be (or is already) placed, a specification (logical) viewpoint, specifying what that system must know and do, and an implementation (physical) viewpoint, specifying in detail the construction of that system. [3]

[1] ISO, RM-ODP [X.900]. www.joaquin.net/RM-ODP/

[2] ANSI/X3/SPARC, DBMS Framework Report, Information Systems, 3, 1978.

[3] Daniels, Modeling with a Sense of Purpose, IEEE Software, 19:1, January 2002.



I'll define how I am using 'abstraction,' just so we can be more exact.

Abstraction: The process of suppressing irrelevant detail to establish a simplified model, or the result of that process.

[RM-ODP 2-6.3 www.joaquin.net/ODP/Part2/6.html#6.3]

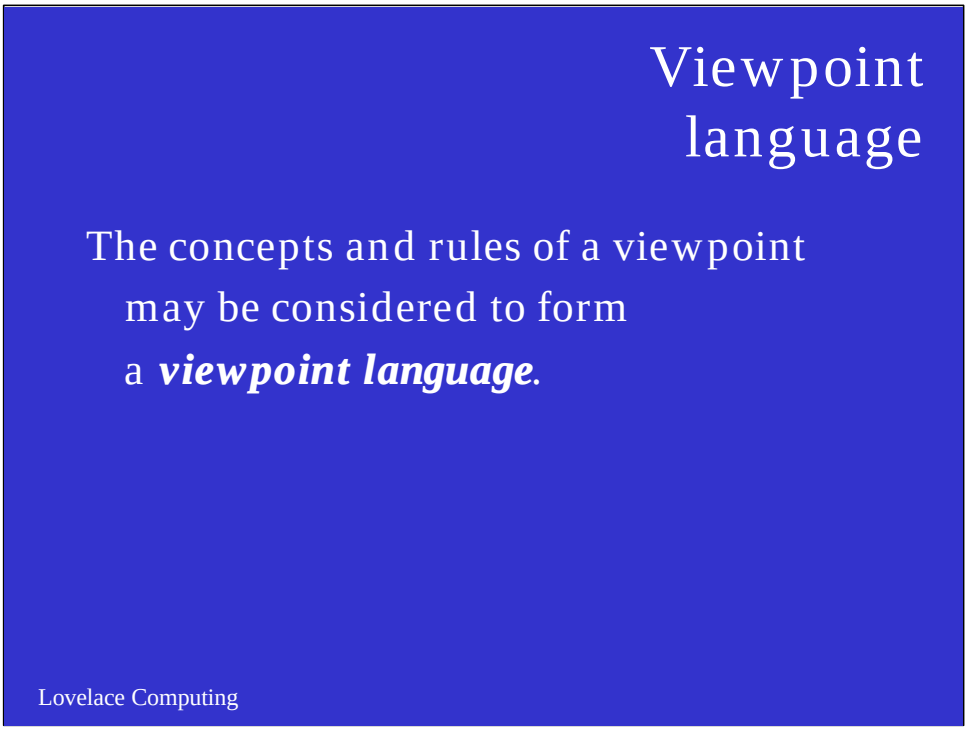
MDA Viewpoints

The Model-Driven Architecture specifies three viewpoints on a system:

- a computation independent viewpoint
- a platform independent viewpoint
- a platform specific viewpoint.

Lovelace Computing

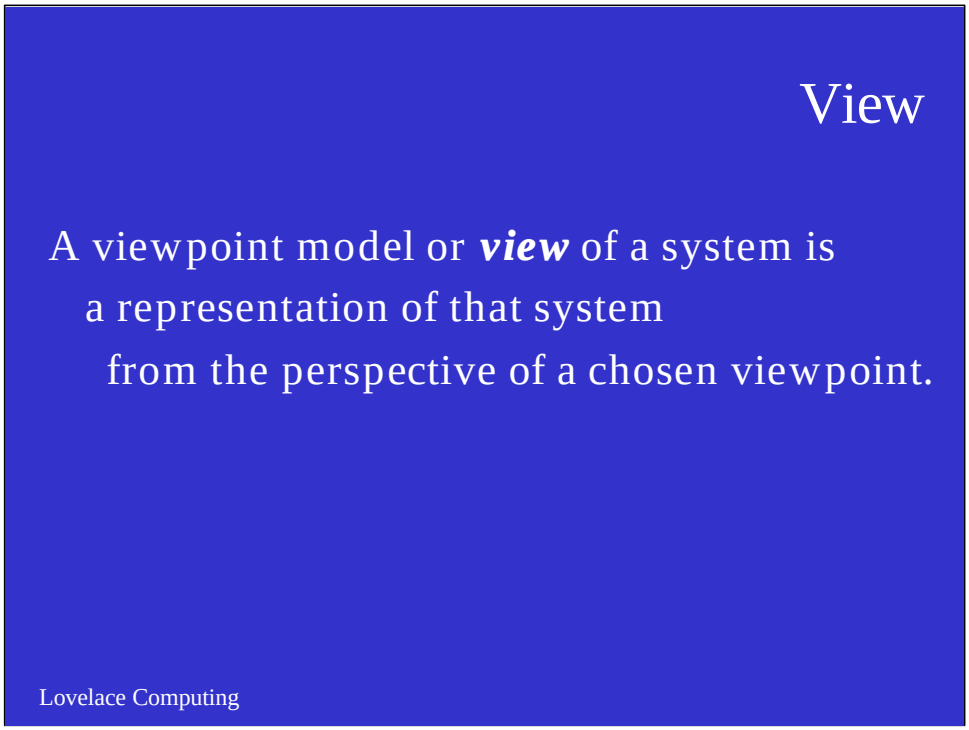
We'll discuss these viewpoints in what follows.



<Viewpoint> language: Definitions of concepts and rules for the specification of an ODP system from the <viewpoint> viewpoint; thus: engineering language: definitions of concepts and rules for the specification of an ODP system from the engineering viewpoint.

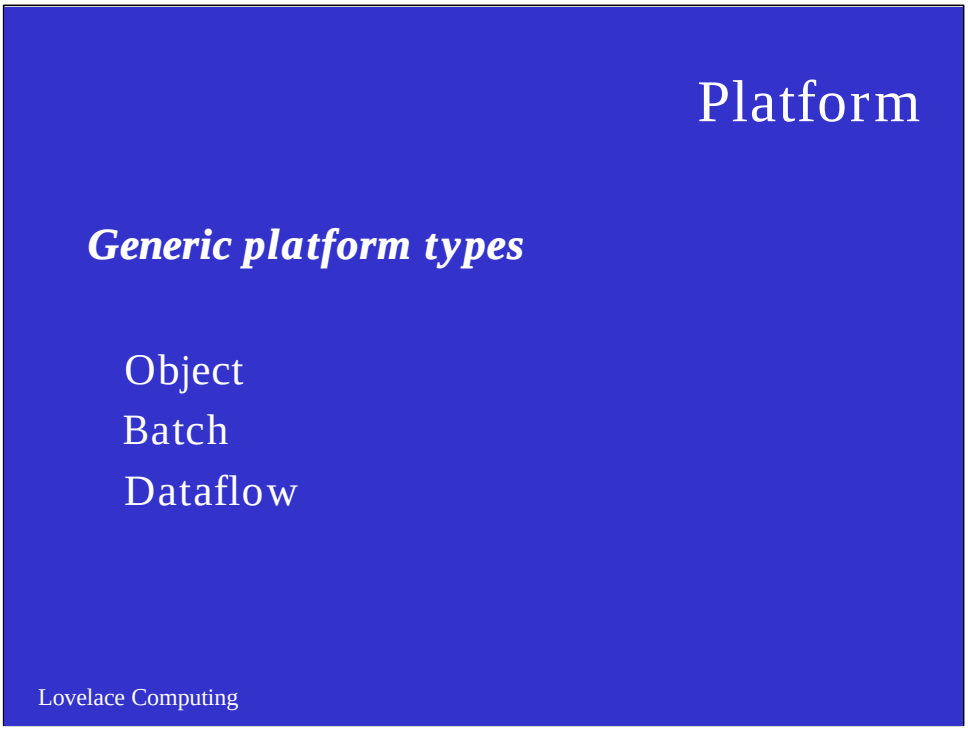
[RM-ODP Part3-4.2.1.1 www.joaquin.net/ODP/Part3/4.html#4.2.1.1]

A modeling language intended for specifying a platform specific model for a certain type of platform can be called a platform specific language.



A viewpoint model or *view* of a system is a representation of that system from the perspective of a chosen viewpoint. The distinction between viewpoint and viewpoint model is important. 'View' is a convenient term for viewpoint model. [4]

[4] IEEE Recommended Practice for Architectural Description of Software-Intensive Systems IEEE Standard 1471-2000.



The Object and Reference Model Subcommittee of the OMG Architecture Board (ORMSC) was careful and deliberate in not defining platform in its draft of the OMG MDA Guide. Here are some examples of kinds of platforms:

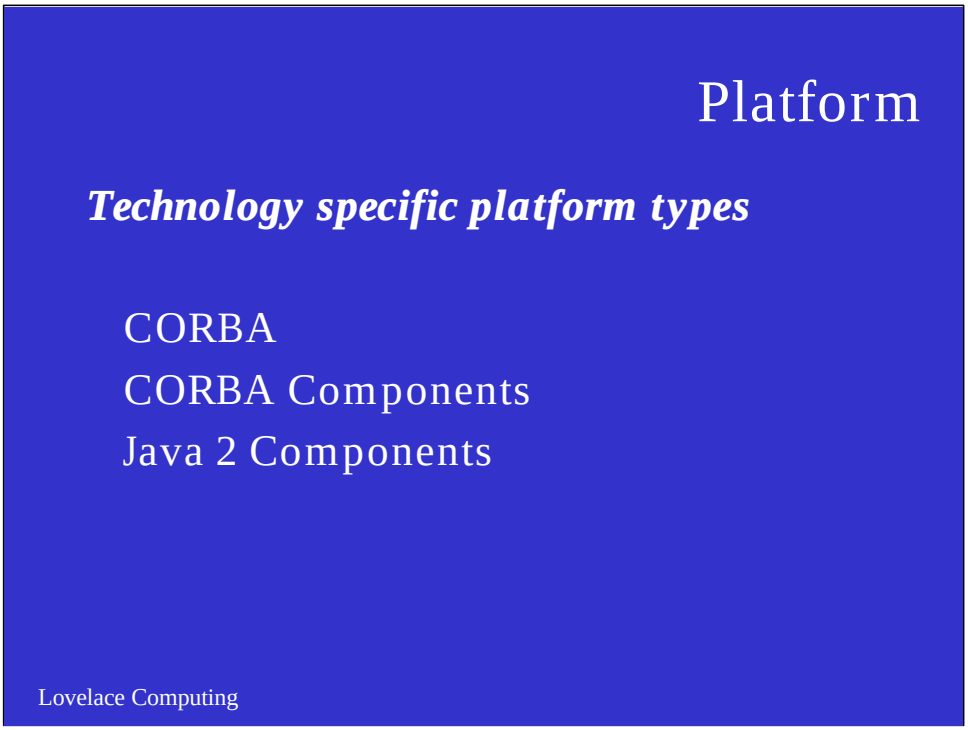
Generic platform types

Object: A platform that supports the familiar architectural style of objects with interfaces, individual requests for services, performance of services in response to those requests, and replies to the requests. [5]

Batch: A platform that supports a series of independent programs that each run to completion before the next starts.

Dataflow: A platform that supports a continuous flow of data between software parts.

[5] Shaw and Garlan, Software Architecture, Prentice Hall ISBN 0-13-182957-2

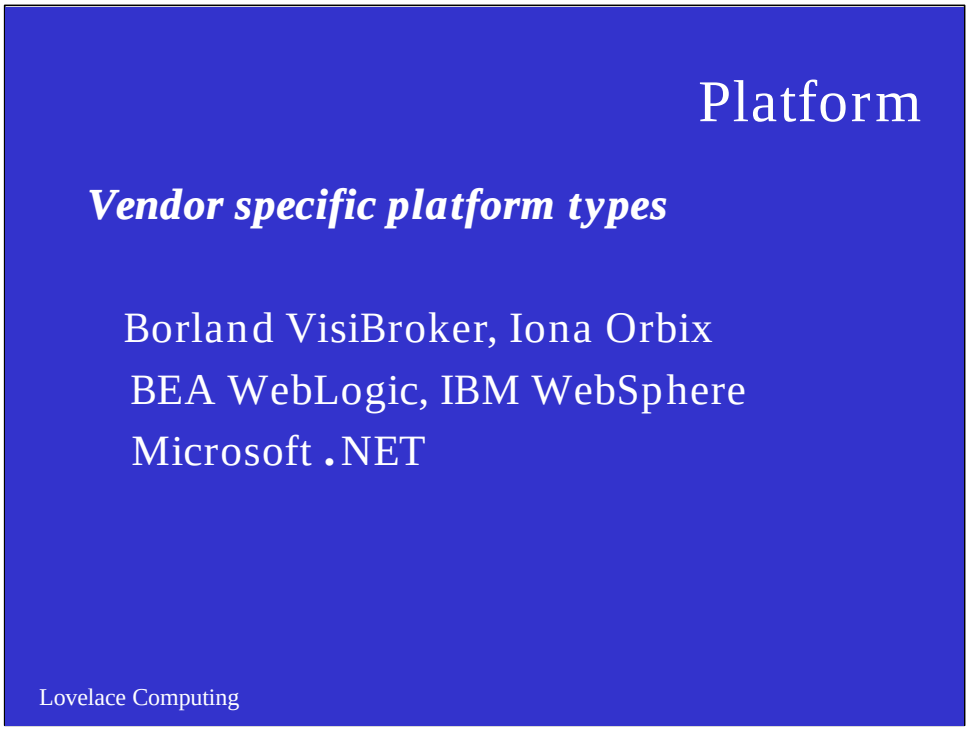


Technology specific platform types

CORBA: An object platform that enables the remote invocation and event architectural styles.

CORBA Components: An object platform that enables a components and containers architectural style. Java 2 Components: Another platform that enables a components and containers style.

Java 2 Components: Another platform that enables a components and containers style.



Vendor specific platform types

CORBA: Iona Orbix, Borland VisiBroker, and many others

Java 2 Components: BEA WebLogic Server, IBM WebSphere software platform, and many others

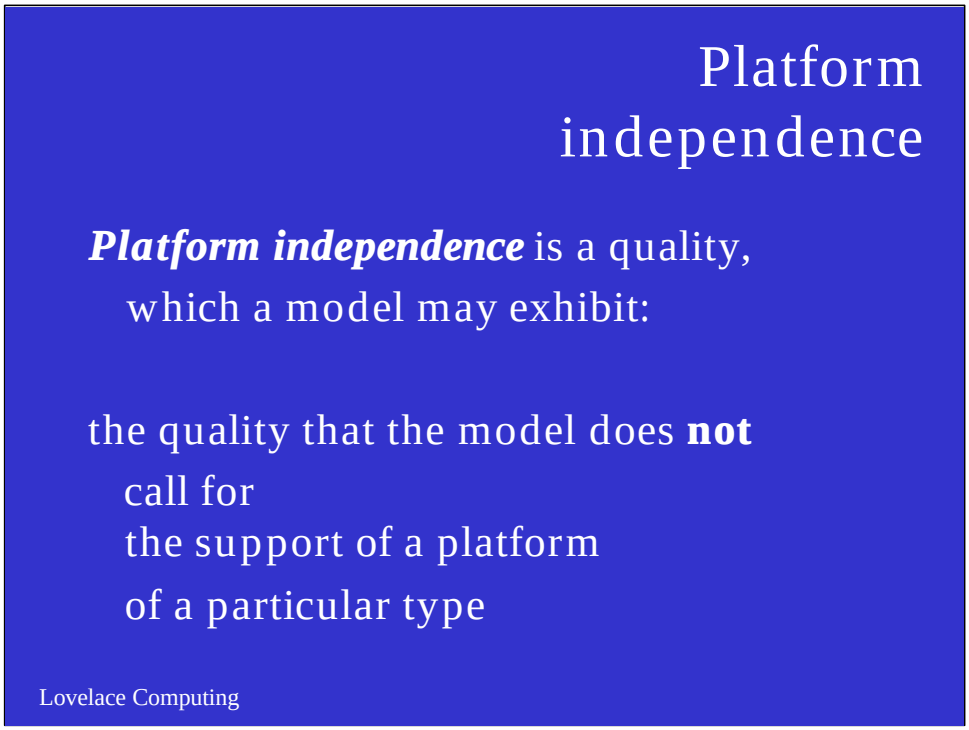
Microsoft .NET

Application

In this presentation, to focus on software, a system will be described as comprising one or more ***applications***, supported by one or more platforms.

Lovelace Computing

Application is just a term of convenience, to distinguish the software being specified from the platform that will support that software.



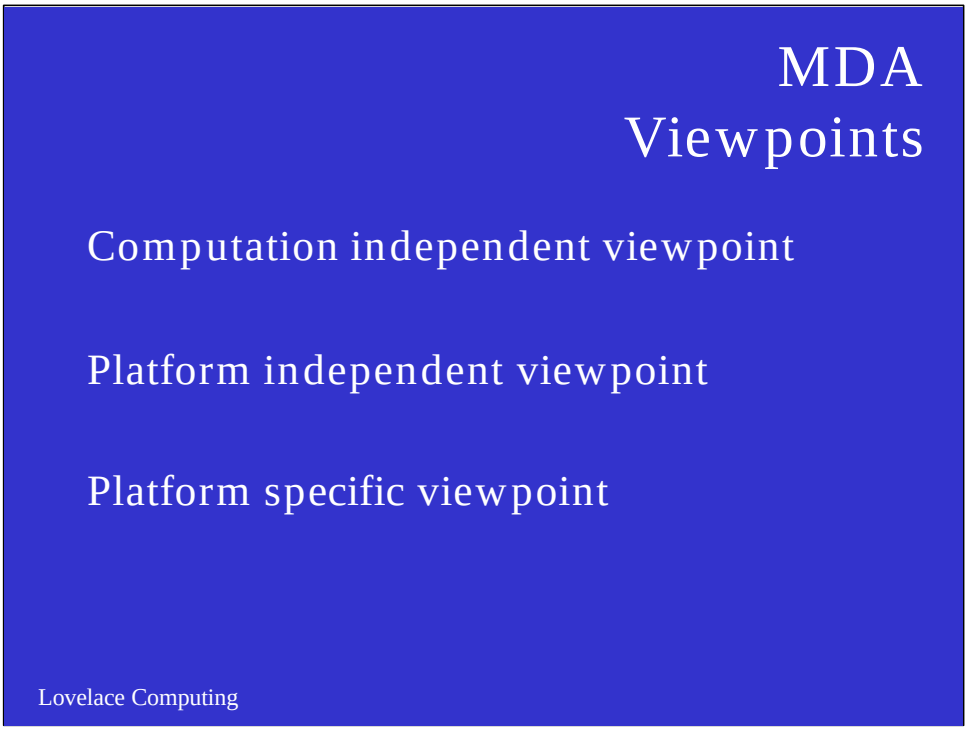
Platform independence is a quality, which a model may exhibit. This is the quality that the model is independent of the features of a platform of a particular type.

Like most qualities, platform independence is a matter of degree. So, one model might only assume availability of features of a very general type of platform, such as remote invocation, while another model might assume the availability a particular set of tools for the CORBA platform. Likewise, one model might might be dependent on a particular type of platform, but only because it assume the availability of one feature of a particular type of platform, while another model might be fully committed to that type of platform.

Questions?

Does anyone have a different take on the concepts discussed so far?

Can everyone go forward with no definition of ‘platform’ **?**



The description of the Model Driven Architecture in the MDA guide is based on the concept, viewpoint.

Remember that a viewpoint is a form of abstraction achieved using a selected set of architectural concepts and structuring rules, in order to focus on particular concerns within a system. So, each of the three MDA viewpoints is a way to prepare a model of a system. The same system is seen from a different viewpoint in different viewpoint models of that system.

Computation independent viewpoint

The ***computation independent viewpoint*** focuses on the system

and its environment.

The details of the structure of the system are hidden or as yet undetermined.

Lovelace Computing

There is some tension in OMG on just what does or should count as a computation independent viewpoint model. That is, on what the concepts and structuring rules are, which define the computation independent viewpoint.

The MDA Guide says: A *computation independent model* is a view of a system from the computation independent viewpoint. A CIM does not show details of the structure of systems. A CIM is sometimes called a domain model and a vocabulary that is familiar to the practitioners of the domain in question is used in its specification.

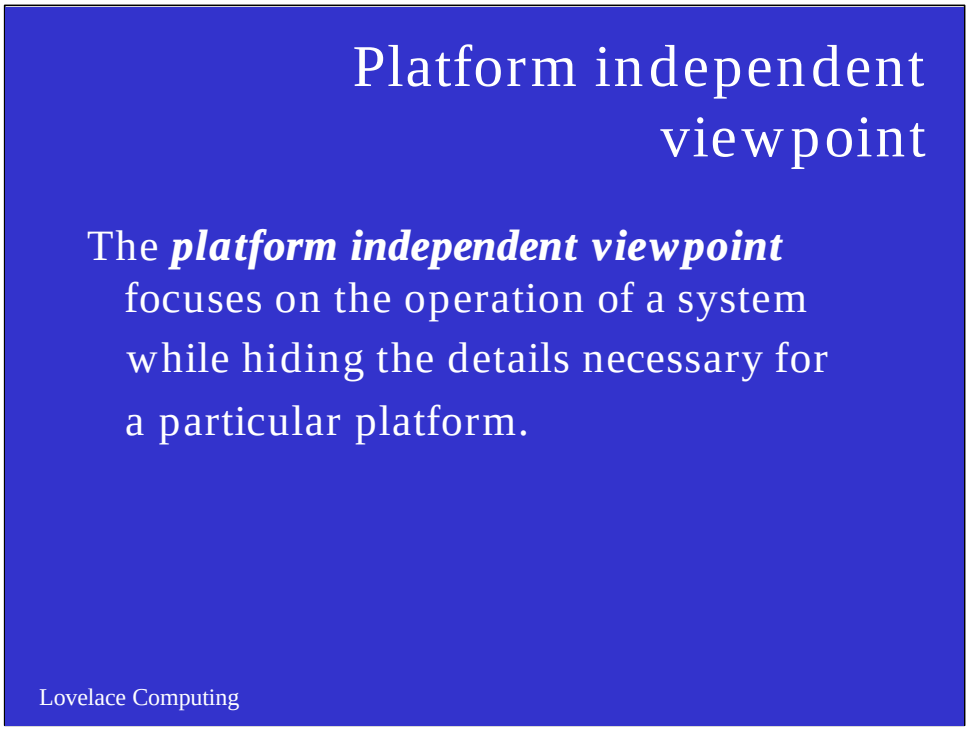
It is assumed that the primary user of the CIM, the domain practitioner, is not knowledgeable about the models or artifacts used to realize the functionality for which the requirements are articulated in the CIM. The CIM plays an important role in bridging the gap between those that are experts about the domain and its requirements on the one hand, and those that are experts of the design and construction of the artifacts that together satisfy the domain requirements, on the other.

The MDA technology adoption document says: The computation independent business model is one in which the Computational (c.f.

RM-ODP computational viewpoint) details are hidden or as yet undetermined.

RM-ODP Computational viewpoint: A viewpoint on an ODP system and its environment which enables distribution through functional decomposition of the system into objects which interact at interfaces.

[RM-ODP 3-4.1.1.3 www.joaquin.net/ODP/Part3/4.html#4.1.1.3]

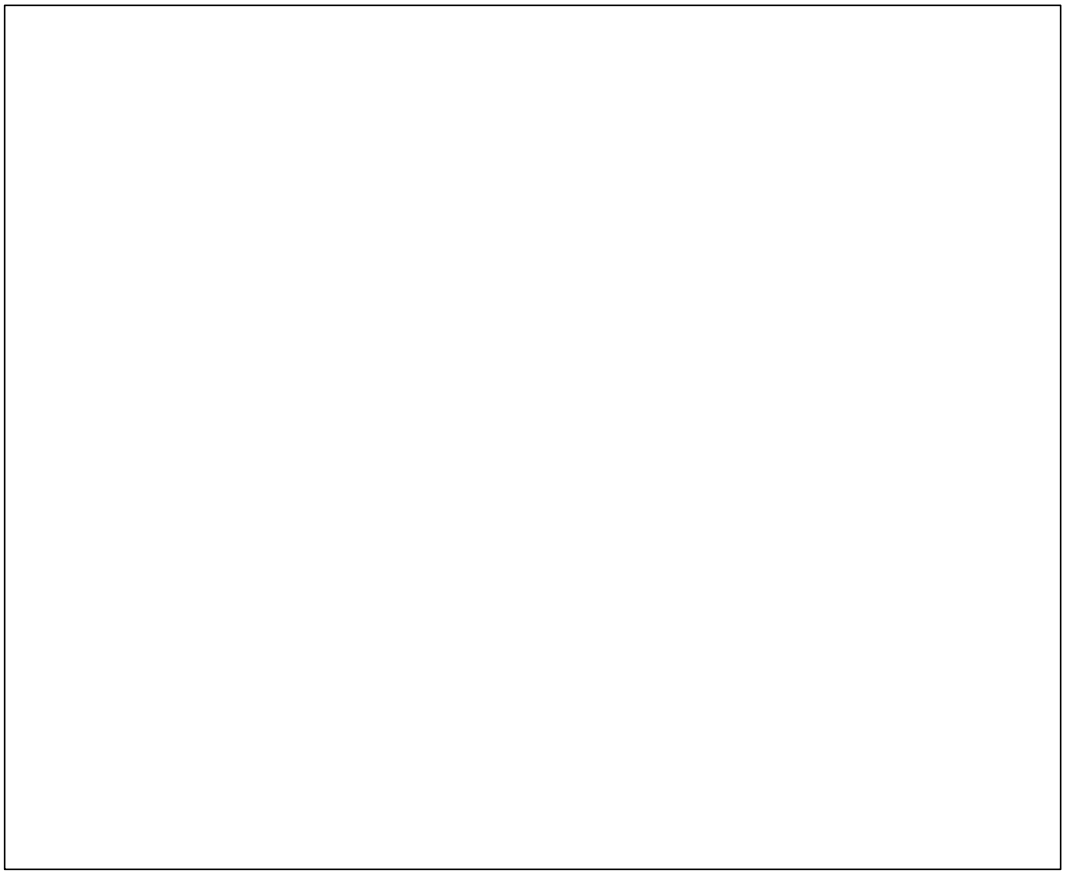


The *platform independent viewpoint* focuses on the operation of a system while hiding the details necessary for a particular platform. A platform independent view shows that part of the complete specification that does not change from one platform to another.

A platform independent view may use a general purpose modeling language, or a language specific to the area in which the system will be used.

Platform independent viewpoint

A platform independent view shows
that part of the complete specification
that does not change
from one platform to another.



Platform specific viewpoint

The ***platform specific viewpoint*** combines the platform independent viewpoint with an additional focus on the detail of the use of a specific platform by a system.

Lovelace Computing

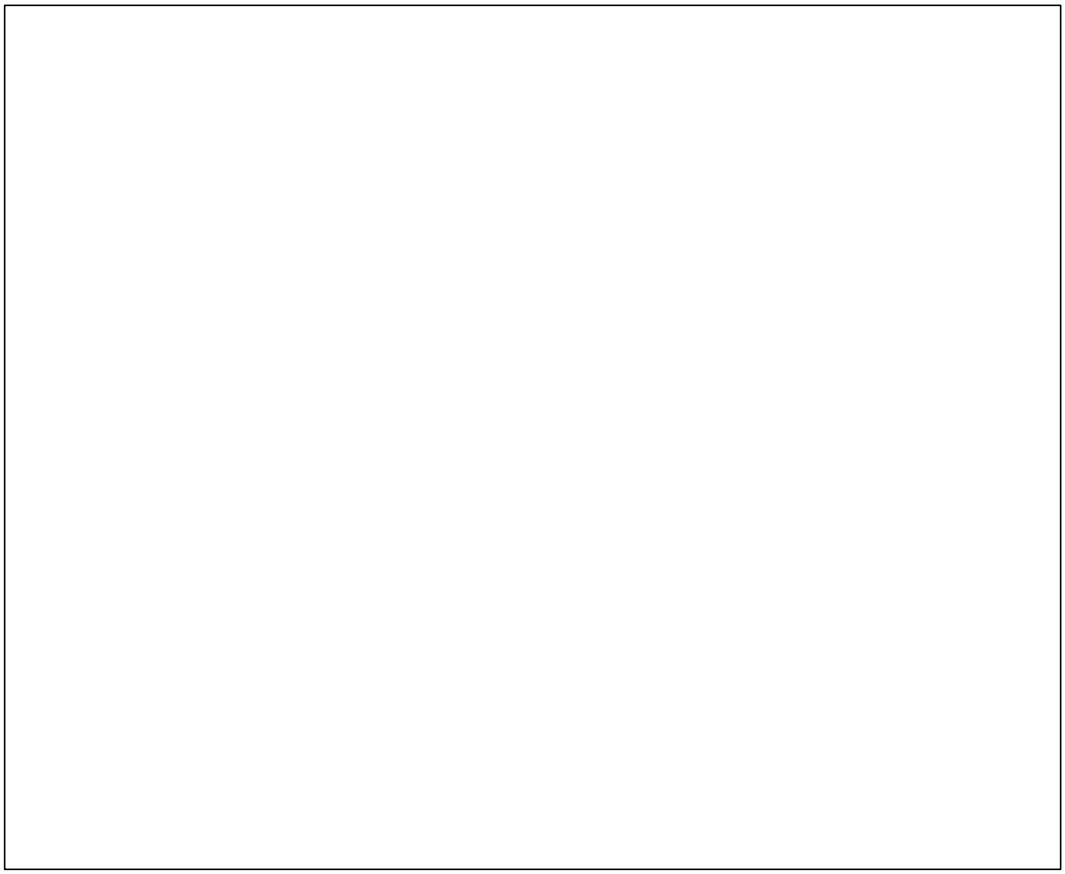
Notice that this means that whatever is represented in a platform independent model is also represented in a corresponding platform specific model.

And notice that this is not the same as to write: whatever ***appears*** in a platform independent model also ***appears*** in a corresponding platform specific model.

The two models may be quite different, but they represent the same things. The platform independent model represents those things in a platform independent way. The platform specific model will usually include more detailed representations and representations of things not represented in the platform independent model.

MDA
model types

- Computation independent model
- Platform independent model
- Platform specific model
- Platform model

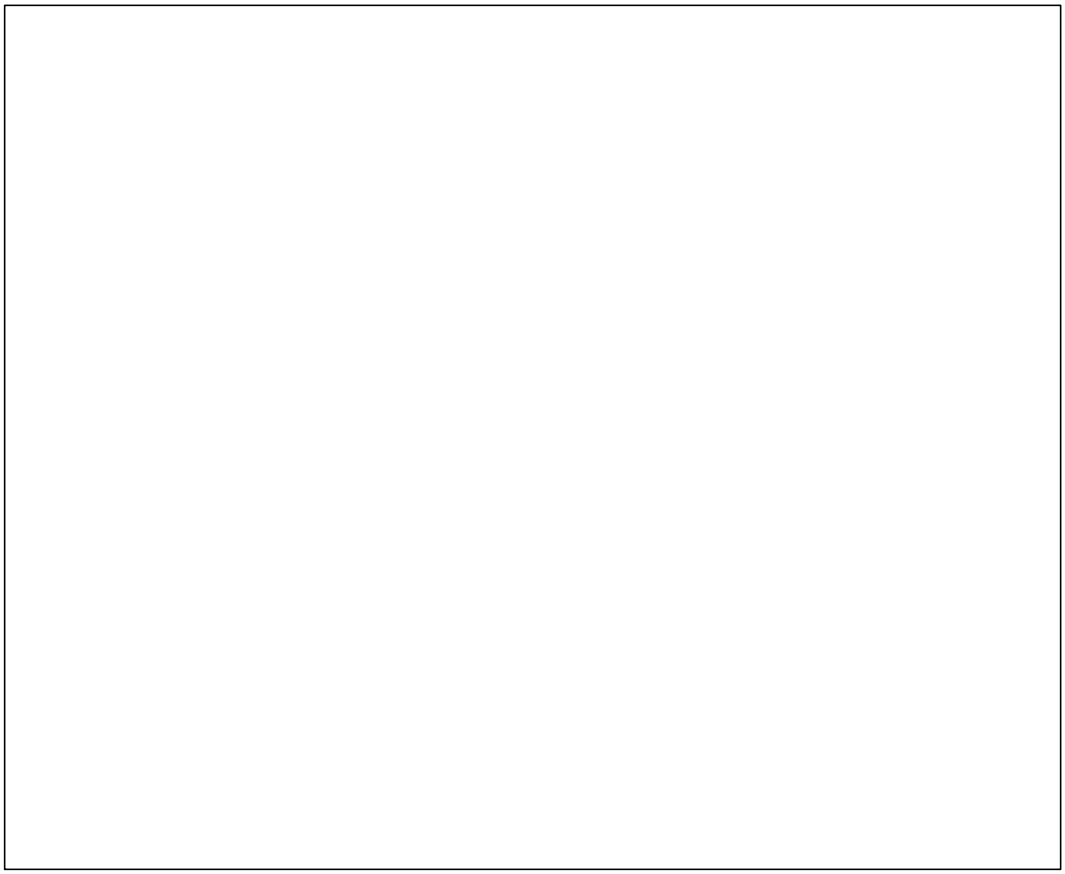


Platform independent model (PIM)

A *platform independent model* is
a view
of a system from the
platform independent viewpoint.

A PIM exhibits platform independence
and is suitable for use with
a number of different platforms
of similar type.

Lovelace Computing

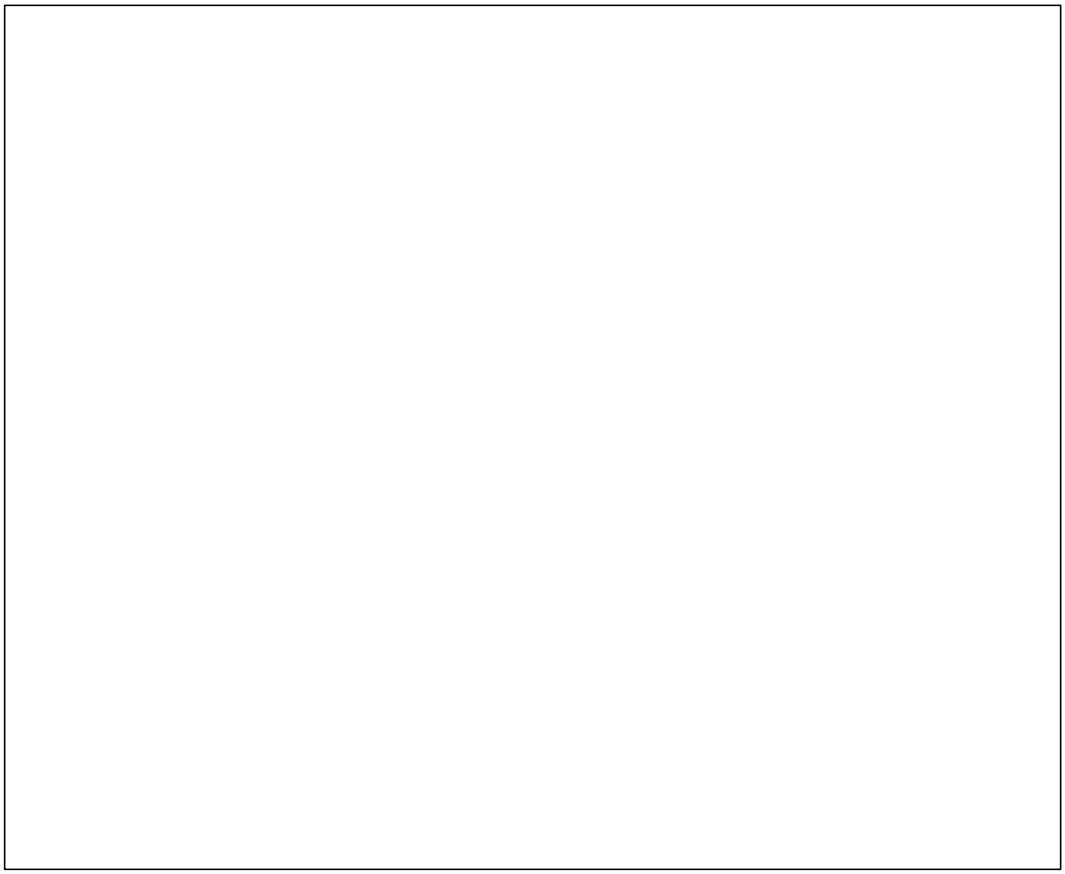


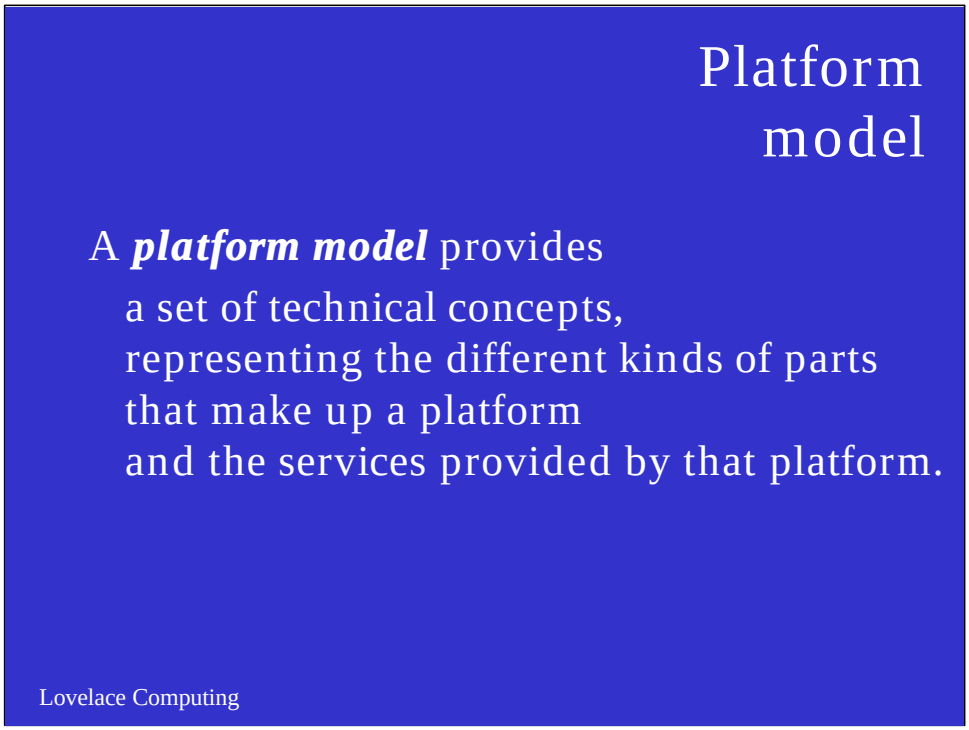
Platform specific model (PSM)

A *platform specific model* is
a view of
a system from the
platform specific viewpoint.

A PSM combines
the specifications in the PIM with
the details that specify how that system
uses a particular type of platform.

Lovelace Computing





A *platform model* provides a set of technical concepts, representing the different kinds of parts that make up a platform and the services provided by that platform. It also provides, for use in a platform specific model, concepts representing the different kinds of elements to be used in specifying the use of the platform by an application.

Example: The CORBA Component Model provides the concepts, EntityComponent, SessionComponent, ProcessComponent, Facet, Receptacle, EventSource, and others. These concepts are used to specify the use of the CORBA Component platform (CCM) by an application.

A platform model also specifies requirements on the connection and use of the parts of the platform, and the connections of an application to the platform.

Example: OMG has specified a model of a portion of the CORBA platform in the UML profile for CORBA. [formal-02-04-01] *This profile provides a language to use when specifying CORBA systems. The stereotypes of the profile can be used as a set of markings.*

A generic platform model can amount to a specification of a particular architectural style.

Platform model

A platform model also provides,
for use in a platform specific model,
concepts representing the different kinds of elements to be used in specifying
the use of the platform by an application.

Lovelace Computing

Platform model

A platform model also specifies requirements on the connection and use of the parts of the platform, and the connections of an application to the platform.

Platform model

Example:

OMG has specified a model of a portion of the CORBA platform in the UML profile for CORBA. [formal-02-04-01]

This profile provides a language to use when specifying CORBA systems. The stereotypes of the profile can be used as a set of markings.

Platform model

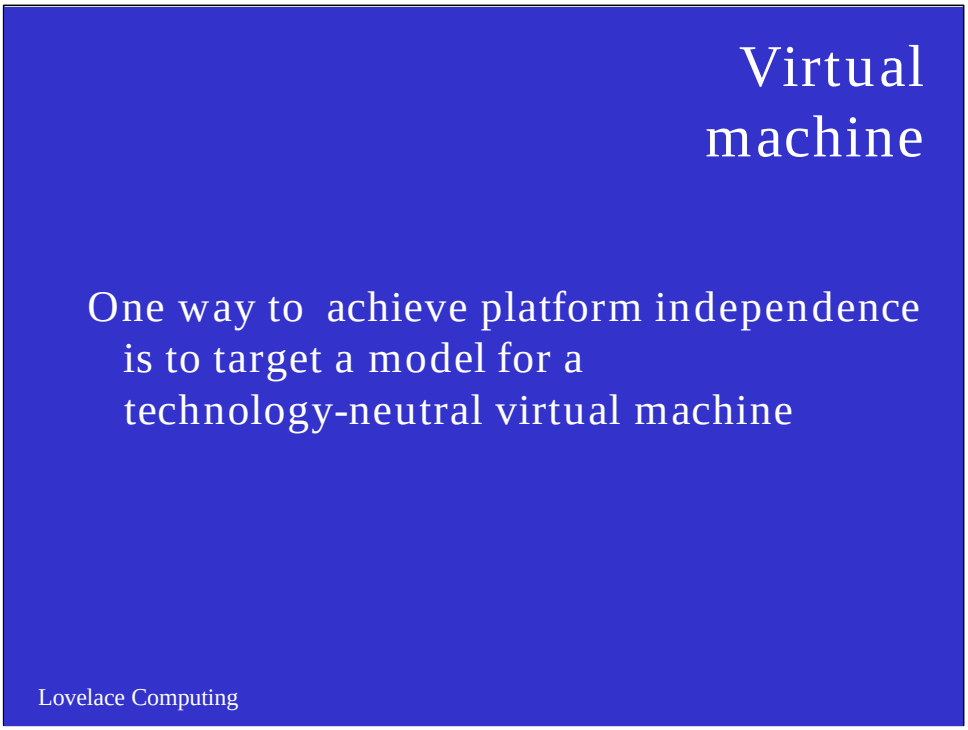
A generic platform model can amount to a specification of a particular architectural style.

Platform model

Example:

The CORBA Component Model provides the concepts, EntityComponent, SessionComponent, ProcessComponent, Facet, Receptacle, EventSource, and others. These concepts are used to specify the use of the CORBA Component platform (CCM) by an application.

Lovelace Computing



A very common technique for achieving platform independence is to target a system model for a technology-neutral virtual machine. A virtual machine is a system that is specified independently of any specific platform and which is realized in platform-specific ways on different platforms. A virtual machine is a platform, and such a model is specific to that platform. But that model is also platform independent with respect to the class of different platforms on which that virtual machine has been implemented. This is because such models are unaffected by the underlying platform and, hence, fully conform to the criterion of platform independence defined in the MDA Guide.

For a PIM based on a virtual machine, transformations are not necessary. Instead, it is the PIM of the virtual machine itself that needs to be transformed to a PSM for a particular platform. When this is done independently of any specific system, the platform specific virtual machine be used with any system targeted to that virtual machine.

Model Transformation

Model transformation is the process of converting one model to another model of the same system.

Lovelace Computing

Model Transformation

Model transformation is the process of converting one model to another model **of the same system**.

Lovelace Computing

Questions?

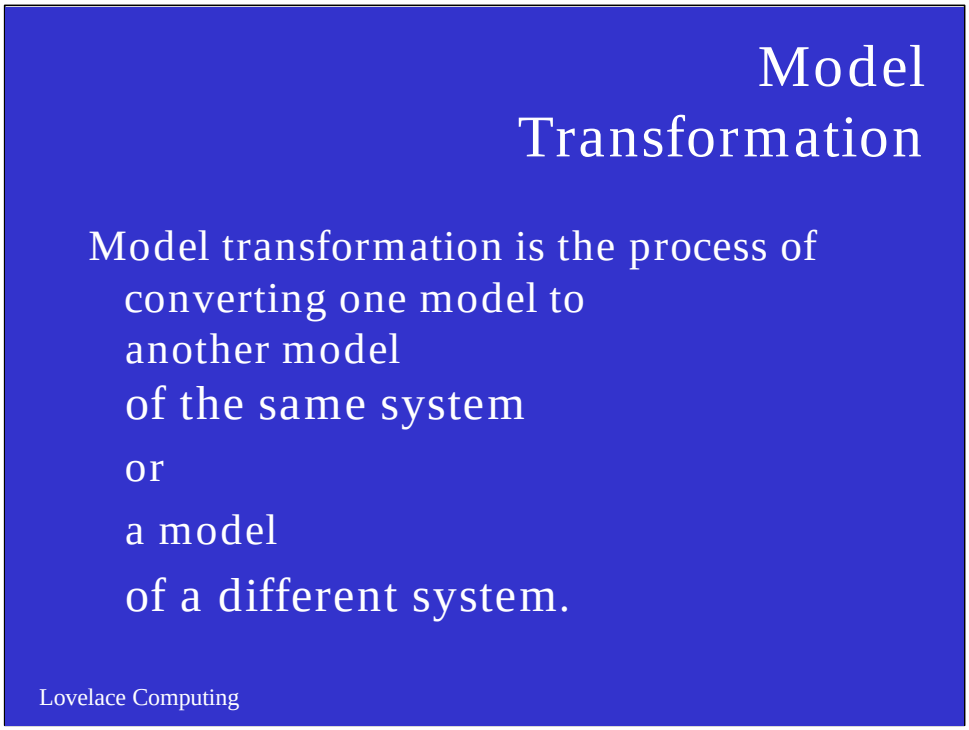
Does anyone have a different take on the concepts discussed so far?

Platform?
Model?
Transformation?

Questions?

Does anyone have a different take on the concepts discussed so far?

… of the same system?



As we will come to discuss, model transformation can be an extremely general concept, and MDA tools provide very general model transformation capabilities.

The point that “of the same system” wants to make is this:

In the straightforward application of MDA to build or modify a system, the transformation from PIM to PSM is about adding platform specific detail about a system to a platform independent specification or description of that system.

If we think about it for a minute, it becomes clear that the relation, same as, does not even apply. Since platform specific detail is hidden in the platform independent viewpoint, it is not possible to determine if the PIM and PSM specify the same system. What is the case is that the PIM specifies a class of equivalent systems for different platoforms; the systems are equivalent in that they correspond to the same PIM.

Here is something that is true, exactly: A PIM specifies a whole class of systems, and the system specified by a PSM produced using that PIM is a member of that class of systems.

But, because of the generality of model transformation, it is not true that any model produced by using a PIM and a transformation is a member of the class of models specified by that PIM. A model transformation may produce a target model that represents a system entirely different from the system represented by the source model.

Example–In a product line architecture, models of different products might be produced from the same starting model using transformations.

Implementation

An implementation is a specification,
which provides all the information needed
to construct a system and
to put it into operation.

Lovelace Computing

Separation of concerns

— specifying a system independently of the platform that supports it
— specifying platforms
— choosing a particular platform for the system
— transforming the specification into one for a particular platform.

Lovelace Computing

Separation of concerns is an old engineering principle. Dijkstra is generally credited for bringing this idea to the attention of software folk. [21]

[21] Dijkstra, A Discipline of Programming, Prentice Hall, 1976

Dijkstra: “I have a small mind and can only comprehend one thing at a time.”

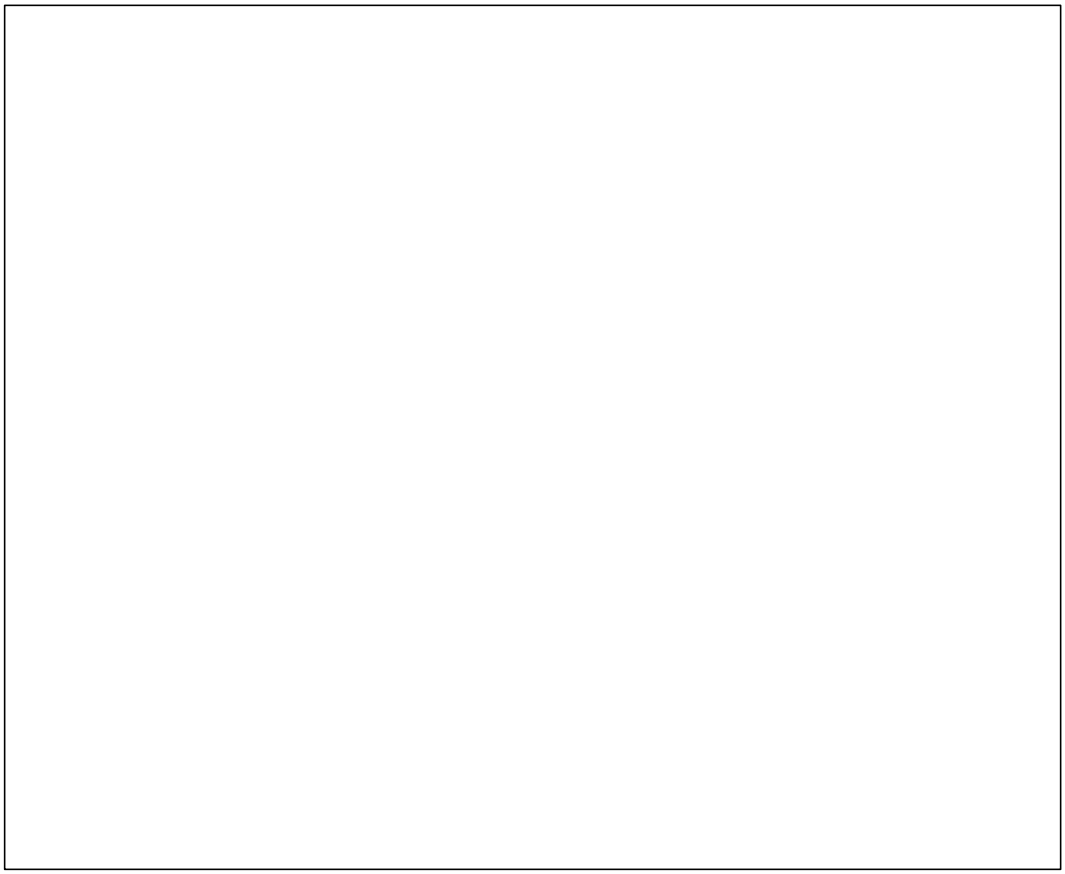
Gries: “When faced with any large task, it is usually best to put aside some of its aspects for a moment and concentrate on others.”

Dijkstra: “Study in depth an aspect of one's subject matter in isolation, for the sake of its own consistency, all the time knowing that

one is occupying oneself with only one of the aspects.”

How MDA is used

- PIM
- Mapping
- Mark
- Transformation
- Record
- PSM



How MDA is used

PIM

Mapping

Mark

Transformation

Record

PSM

Lovelace Computing

Model

Domain model

Computation independent model

Platform independent model

Lovelace Computing

Domain model

— describes the situation in which a system will be used

— may hide much or all information about the use of automated data processing systems

—not only as an aid to understanding, but also a source of shared vocabulary

Lovelace Computing

A model that describes the situation in which a system will be used provides a valuable starting point. Such a model is sometimes called a domain model or a business model. It may hide much or all information about the use of automated data processing systems.

It is useful, not only as an aid to understanding a problem, but also as a source of a shared vocabulary for use in other models.

Computation independent model

— the system in the environment in which it will operate

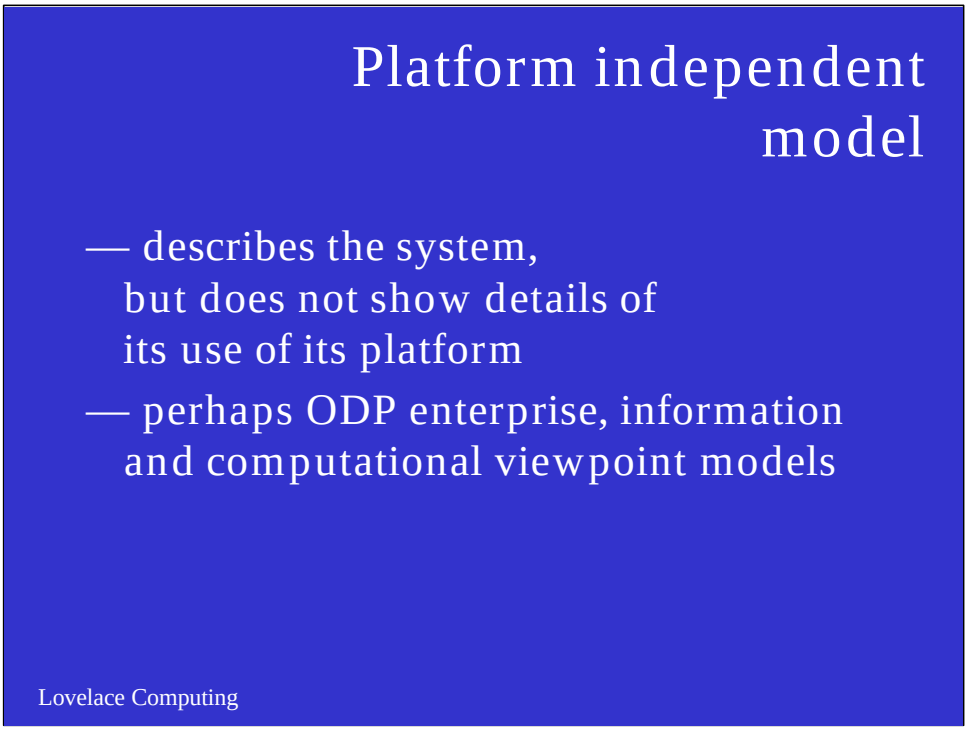
— independent of how the system is implemented.

—perhaps ODP enterprise and information viewpoint models

Lovelace Computing

If a model of a system is prepared showing the system in the environment in which it will operate, that model will help to understand exactly what the system is to do. A model using the computation independent viewpoint is independent of how the system is implemented.

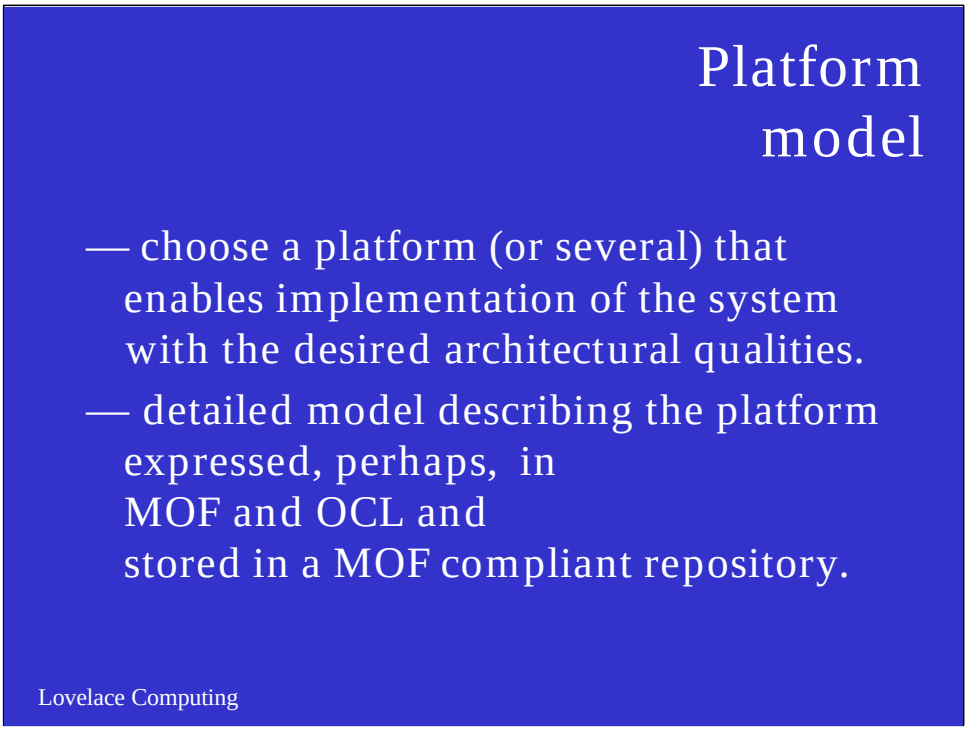
A computation independent model might consist of two UML models, from the ODP enterprise and information viewpoints. It might include several models from these viewpoints, some providing more detail than others, or focusing on particular concerns of a viewpoint.



A platform independent model, a PIM, is built. It describes the system, but does not show details of its use of its platform.

A PIM might consist of enterprise, information and computational ODP viewpoint specifications.

Though independent of some class of platforms, a platform independent model will be suited for a particular architectural style, or several.



The architect will then choose a platform (or several) that enables implementation of the system with the desired architectural qualities.

The architect will have at hand a model of that platform. Often, at present, this model is in the form of software and hardware manuals or is even in the architect's head. MDA will be based on detailed platform models, for example, models expressed in MOF and OCL, and stored in a MOF compliant repository.

How MDA is used

PIM

Mapping

Mark

Transformation

Record

PSM

Lovelace Computing

An MDA mapping provides specifications for transformation of a PIM into a PSM for a particular platform. The platform model will determine the nature of the mapping.

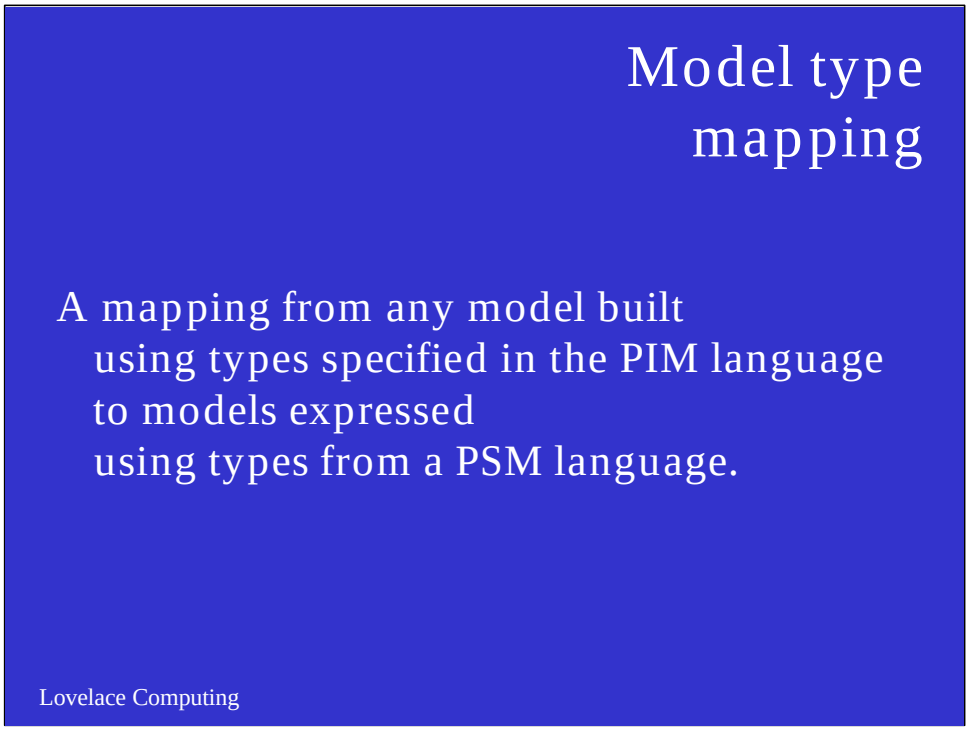
Examples

A platform model for EJB includes the Home and RemoteInterface as well as Bean classes and Container Managed Persistence.

Two examples, illustrating different approaches:

Example 1: An EDOC ECA PIM contains attributes which indicate whether an Entity in that model is managed or not, and whether it is remote or not. A mapping from ECA to EJB will state that every managed ECA entity will result in a Home class, and that every remoteable ECA entity will result in a RemoteInterface. Marks associated with the mapping (with required parameter values) are supplied by an architect during the mapping process to indicate the style of EJB persistent storage to be used for each ECA entity, as no information about this concept is stored in the PIM.

Example 2: A UML PIM to EJB mapping provides marks to be used to guide the PIM to PSM transformation. It also includes templates or patterns for code generation and for configuration of a server. Marking a UML class with the Session mark results in the transformation of that class according to the mapping into a session bean and other supporting classes.



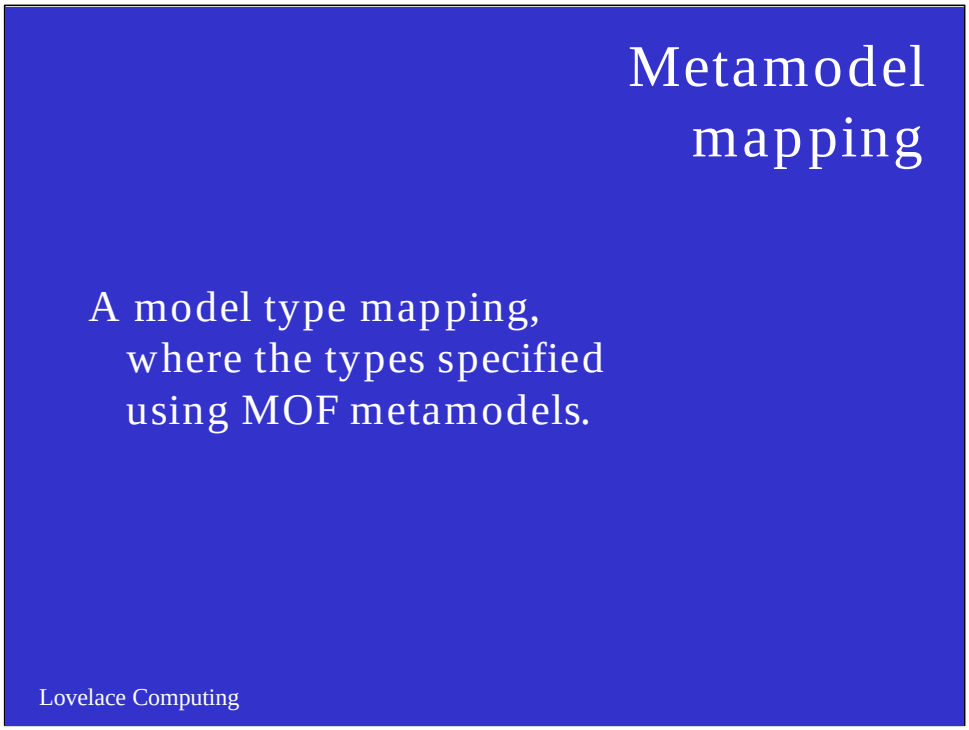
A model type mapping specifies a mapping from any model built using types specified in the PIM language to models expressed using types from a PSM language.

A PIM is prepared using a platform independent model of types. The architect chooses types specified by that model to build the PIM, according to the requirements of the application. One or more model mappings each specify a mapping from elements of the platform independent types to platform specific types. These mappings may also specify mapping rules in terms of the instance values to be found in models expressed in the PIM language.

Example: If the attribute sharable of class, Entity, is ***`true`*** *for a particular PIM model instance of type, Entity, then map to an EJB Entity, otherwise map to a Java Class.*

These kinds of rules may also map things according to patterns of type usages in the PIM.

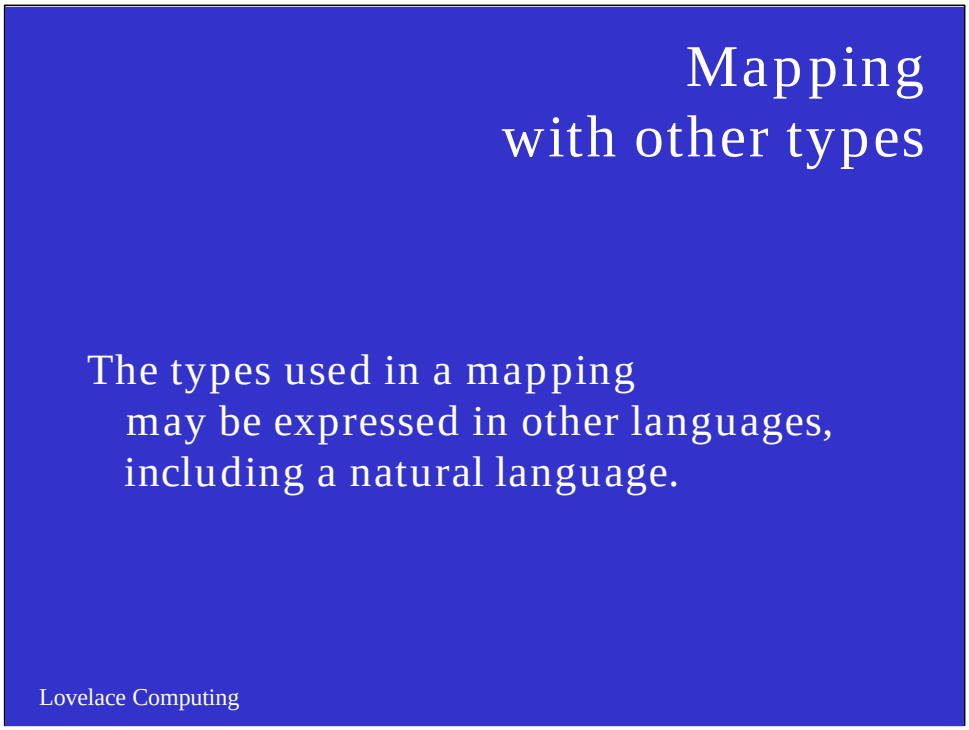
Example: If pattern exists where an instance of class, Entity, has a manages association to an instance of class, Document, whose attribute, persistent, is set, then map that instance to an EJB Entity which manages whatever is mapped from the instance of Document instance identified by the pattern.



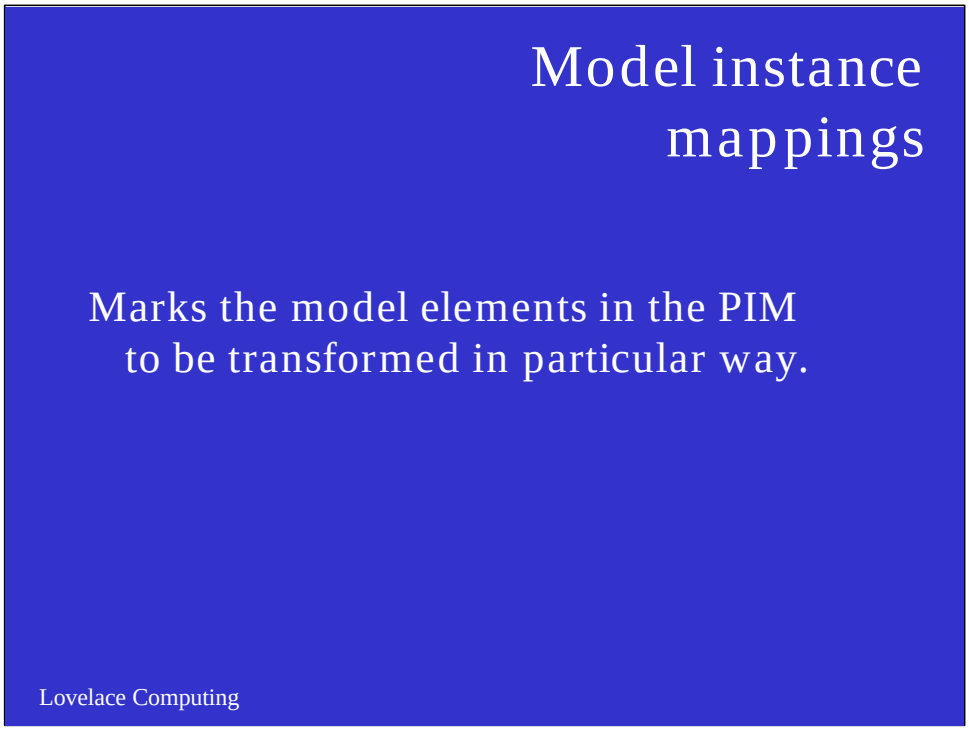
A metamodel mapping is a specific example of a model type mapping, where the types of model elements in the PIM and the PSM are both specified as MOF metamodels. In this case the mapping gives rules and/or algorithms expressed in terms of all instances of types in the metamodel specifying the PIM language resulting in the generation of instances of types in the metamodel specifying the PSM language(s).

Notice that we have a different meaning of 'type' here. In a model type mapping the types are in the language of the model; in a metamodel mapping the types are from a metamodel.

If the previous paragraph seems muddled or hard to follow, or off base, that is one more symptom of the meta-muddle we find ourselves in.

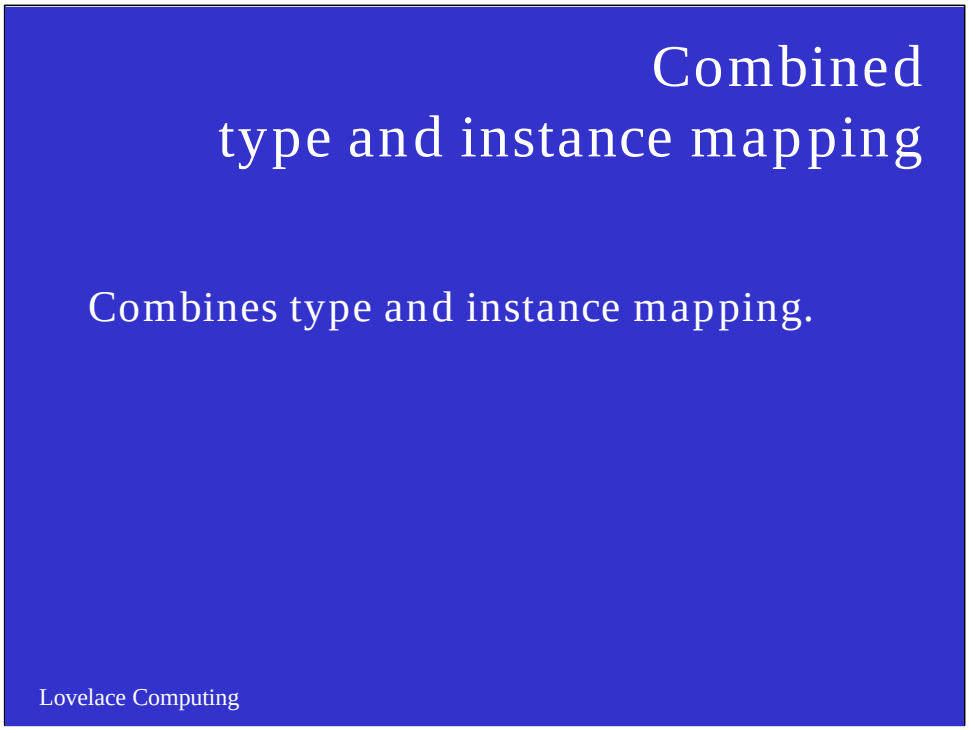


The types available to model the PSM (or even the PIM) may not be specified as a MOF metamodel. For example, the CORBA IDL language provides for the expression of types available in CORBA PSMs. In this case mappings can be expressed as transformations of instances of types in the PIM, into instances of types in the PSM expressed in other languages, including natural language.



Another approach to mapping models is to identify model elements in the PIM which should be transformed in particular way, given the choice of a specific platform for the PSM.

Model instance mappings will use marks. We'll discuss these shortly.



Most mappings, however, will consist of some combination of the above approaches.

A model type mapping is only capable of expressing transformations in terms of rules about things of one type in the PIM resulting in the generation of some thing(s) of some (one or more) type(s) in the PSM. However, without the ability for the architect to also mark the model with additional information for use by the transformation, the mapping will be deterministic, and will rely wholly on Platform Independent information to generate the PSM. Rules in the mapping will often specify that certain types in the PIM must be marked with one of a set of marks in order that the PSM will have the right non-functional or stylistic characteristics, which cannot be determined from information in the PIM.

Likewise, every transformation of model instances has implicit type constraints which the architect marking the model must obey in order for the transformation to make sense. For example, marking an Association End in a UML model with the mark, 'Entity,' makes no sense, whereas marking it with the mark, 'RMI navigable,' does. Implicitly each type of model element in the PIM is only suitable for certain marks, which indicate what type of model element will be generated in the PSM. Transformations based on marking instances will either explicitly state which marks are suitable for which types in the PIM, or these type constraints will be implicitly understood by the user of the marks.

How MDA is used

PIM

Mapping

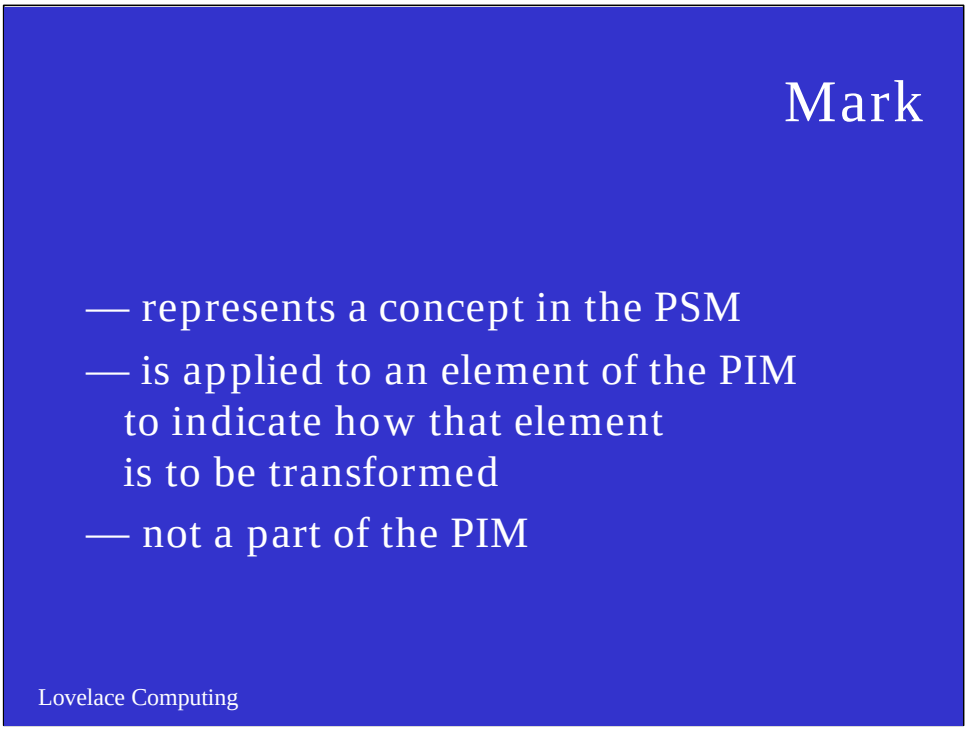
Mark

Transformation

Record

PSM

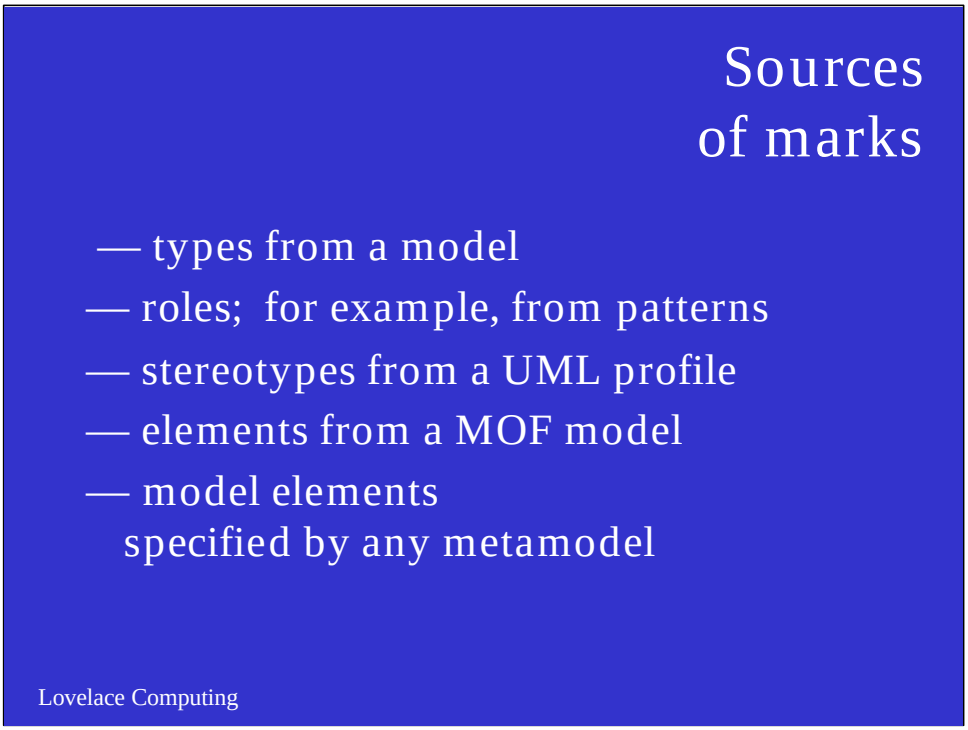
Lovelace Computing



Many model mappings will define marks. A mark represents a concept in the PSM, and is applied to an element of the PIM, to indicate how that element is to be transformed.

The marks, being platform specific, are not a part of the platform independent model. The architect takes the platform independent model and marks it for use on a particular platform. The marked PIM is then used to prepare a platform specific model for that platform.

The marks can be thought of as being applied to a transparent layer placed over the model.



Marks may come from different sources. These include:

— types from a model, specified by classes, associations, or other model elements

— roles from a model, for example, from patterns

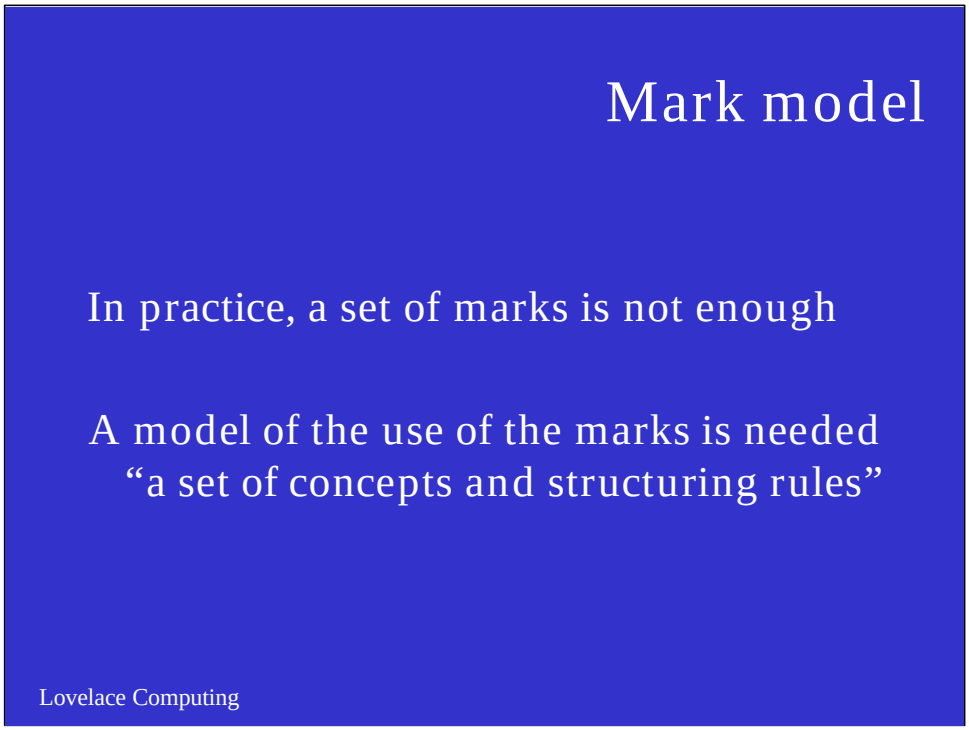
— stereotypes from a UML profile

— elements from a MOF model

— model elements specified by any metamodel

Example: Entity is a mark that can be applied to classes or objects in a PIM; this mark indicates that the Entity template of the mapping will be used in transforming that PIM to a PSM.

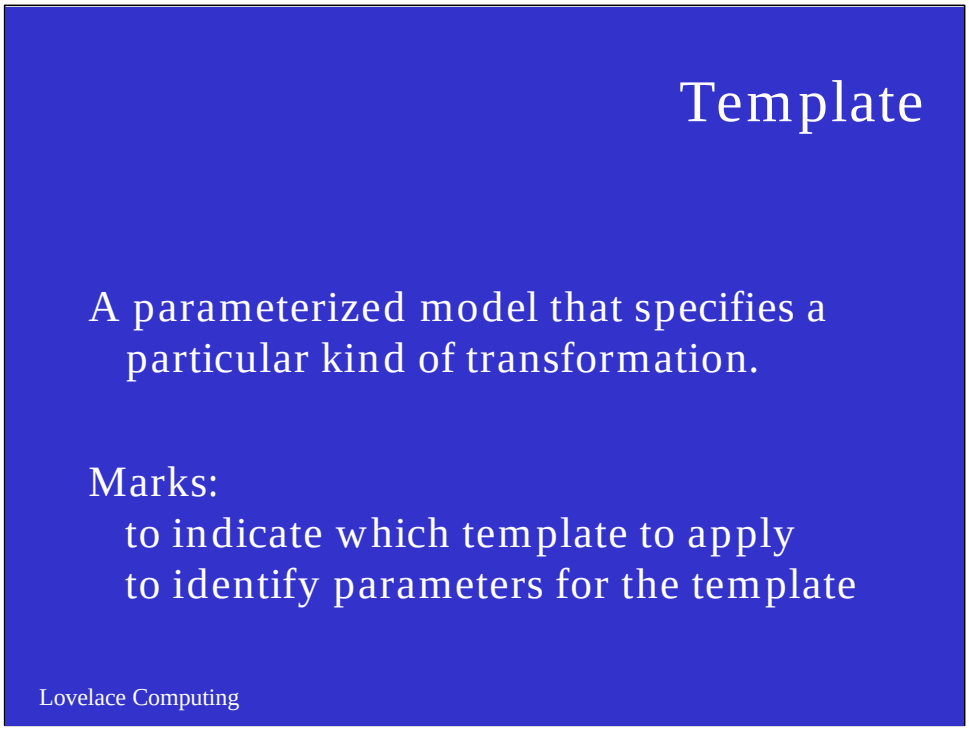
Marks may also specify quality of service requirements on the implementation. That is, instead of indicating the target of a transformation, a mark may instead simply provide a requirement on the target. The transformation will then choose a target appropriate to that requirement.



In order for marks to be properly used, they may need to be structured, constrained or modeled. For example a set of marks indicating mutually exclusive alternative mappings for a concept need to be grouped, so that an architect marking a model knows what the choices are, and that more than one of these marks cannot be applied to the same model element.

Some marks, especially those that indicate quality of service requirements, may need parameters. For example, a mark, 'Supports simultaneous connections,' may require a parameter to indicate an upper bound on the number of connections that need to be supported, or even several parameters giving details for timeouts or connection policy.

A set of marks, instead of being supplied by a mapping, may be specified by a mark model, which is independent of any particular mapping. Such a set of marks can be used with different mappings. A set of marks may also be supplied along with a UML profile; several different mappings might be supplied with that profile.



A mapping may also include templates, which are. These templates are like design patterns, but may include much more specific specifications to guide the transformation.

Templates can be used in rules for transforming a pattern of model elements in a model type mapping into another pattern of model elements.

A set of marks can be associated with a template to indicate instances in a model which should be transformed according to the template. Other marks can be used to indicate which values in a model fill the parameters in the template. This allows values in the source model to be copied into the target model, and modified if necessary.

Example: A CORBA Component mapping might include an Entity template, which specifies that an object in the platform independent model, which is marked, Entity, corresponds, in a platform specific model, to two objects, of types HomeInterface and EntityComponent, with certain connections between those objects.

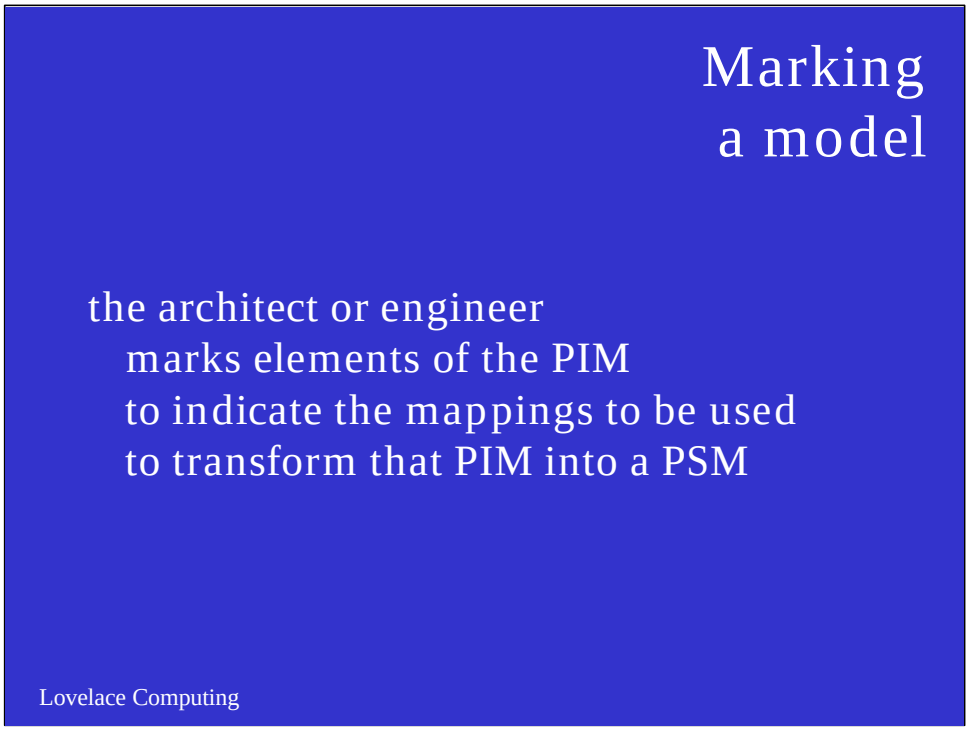
Example: A CORBA mapping might provide that a client object be prepared for a range of CORBA non-standard system exceptions or standard user exceptions and include the necessary exception handling in each case.

Example: A mapping from the EAI metamodel to a COBOL Connector implementation design might identify a template with an Adapter associated with a Connector which has certain attributes as a pattern that is directly mapped to a certain Connector type.

A mapping is specified using some language to describe a transformation of one model to another. The description may be in natural language, an algorithm in an action language, or in a model mapping language.

Model mapping languages are an area for MDA technology adoptions. The current MOF Query / View / Transformation RFP requests technology submissions suited to the specification of metamodel mappings.

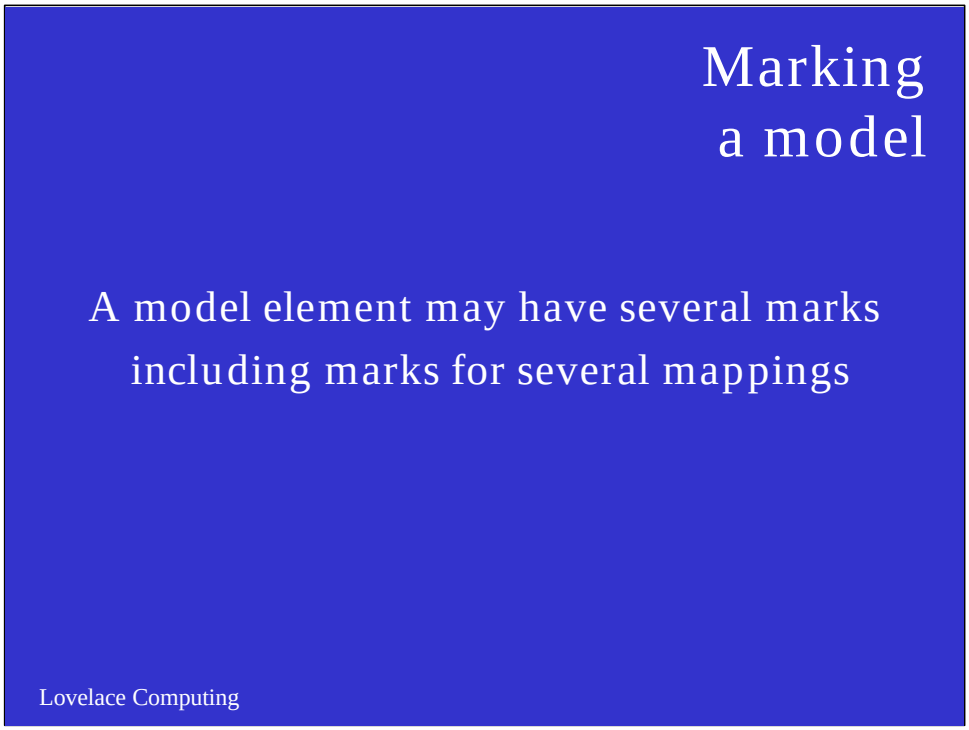
A desirable quality of a mapping language is portability. This enables use of a mapping with different tools.



In model instance mappings the architect marks elements of the PIM to indicate the mappings to be used to transform that PIM into a PSM.

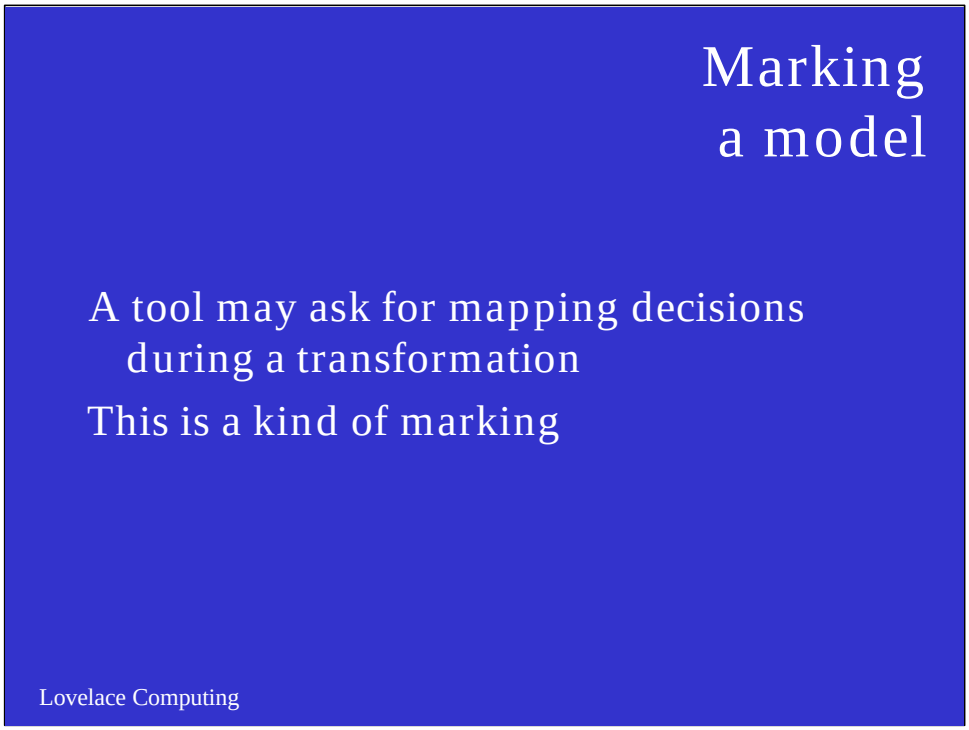
In one simple case, a PIM element is marked once, indicating that a certain mapping is to be used to transform that element into one or more elements in the PSM.

In a more general case, several PIM elements are marked to indicate their roles in some mapping. This mapping is then used to transform those PIM elements into some different set of PSM elements, perhaps quite different in appearance.

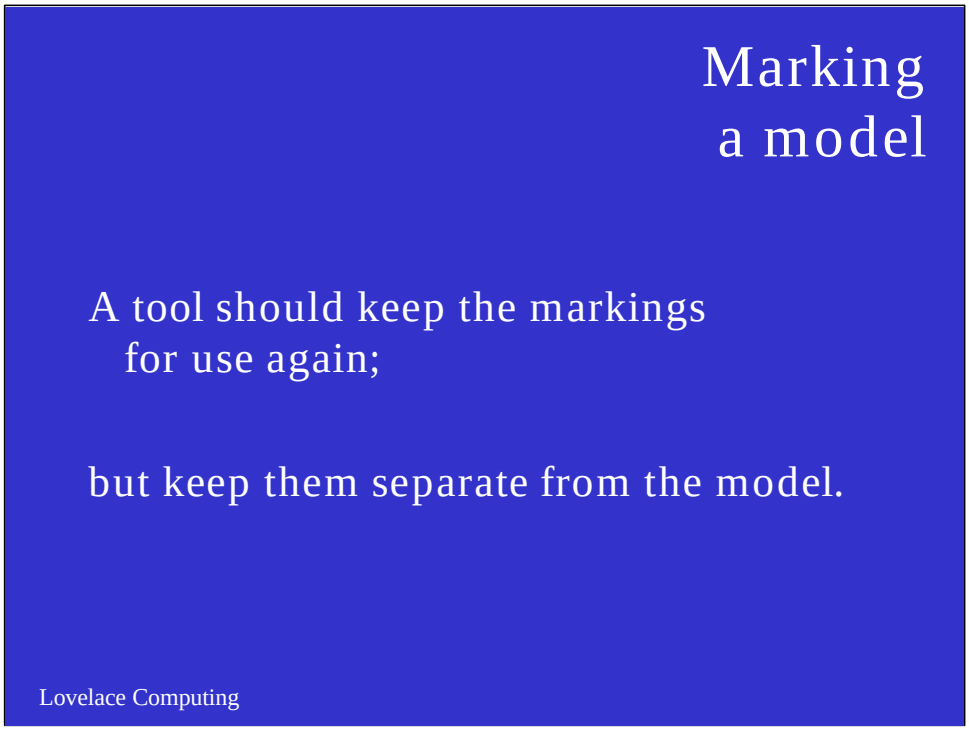


An element of the PIM may be marked several times, with *mark*s from different mappings; this indicates that the element plays a role in more than one mapping. When an element is marked in this way, it will be transformed according to each of the mappings; the result may be additional features of the resulting element(s) as well as additional resulting elements in the PSM.

Example: Entity is a mark in one mapping that can be applied to classes or objects in a PIM; this mark indicates that the Entity template of the mapping will be used in transforming that PIM to a PSM. Auditable is a mark in another mapping; this mark indicates that changes to an object will be recorded in a write only file. When both mappings are applied, an object marked with entity and auditable is transformed according to the Entity template of the first mapping and with a capability to detect and record changes.



In model type transformations a mapping description, specified in terms of rules and / or algorithms is applied to a model of the type that the mapping is designed for. All rules and algorithms which operate on type information automatically generate a target model, but the transformation tool asks a user for mapping decisions in the course of transformation where a rule specifies that information not available in the source model is required, and records those decisions as marking of the PIM.



Model markings can be stored and subsequent transformations may use these marking, asking only for additional decisions required by additions or changes to the model.

How MDA is used

PIM

Mapping

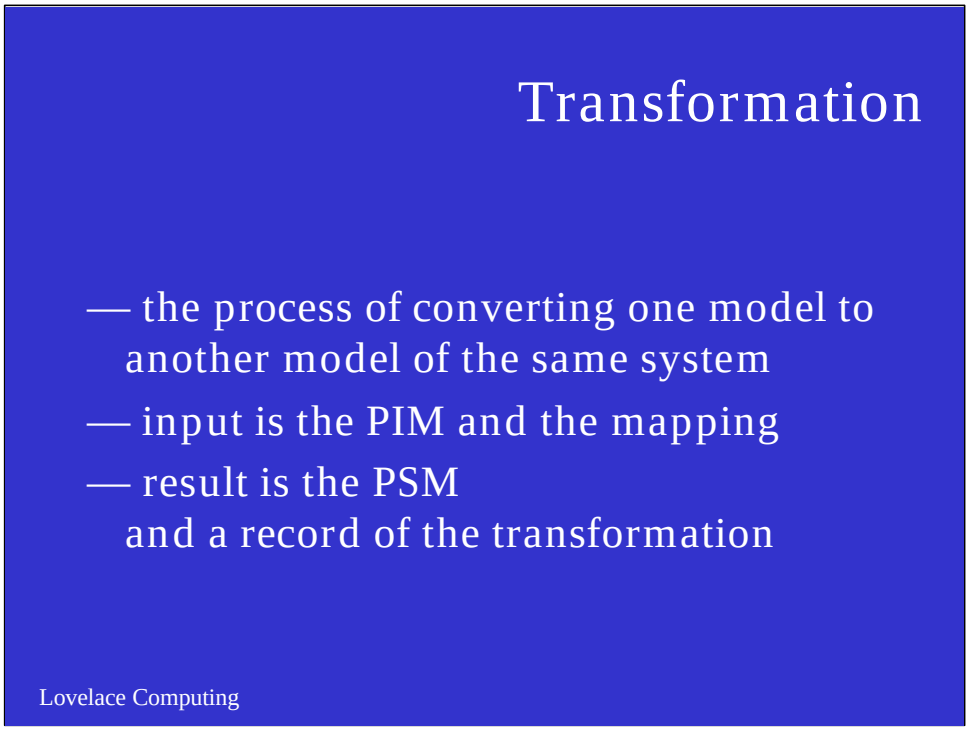
Mark

Transformation

Record

PSM

Lovelace Computing



The next step is to take the marked PIM and transform it into a PSM. This can be done manually, with computer assistance, or automatically.

Model transformation is the process of converting one model to another model of the same system. The input to the transformation is the marked PIM and the mapping. The result is the PSM and the record of transformation.

Using model type mapping, transformation takes any PIM specified using one model and, following the mapping, produces a PSM specified using another model.

Using model instance mapping, transformation takes a marked PIM and, following the mappings, as indicated by the marks, produce a PSM.

Example:

A platform independent model of a securities trading system (a PIM) is transformed for the CORBA component platform. The result of the transformation is a model of that system specific to the CORBA component platform (a PSM) and a record of transformation showing the correspondences between the two models.

Direct to code

— transform a PIM directly to code, without producing a PSM

— or also produce a PSM, for use in understanding or debugging that code

Lovelace Computing

In some cases, a tool will transform a PIM directly to deployable code, without producing a PSM. Such a tool might also produce a PSM, for use in understanding or debugging that code.

How MDA is used

PIM

Mapping

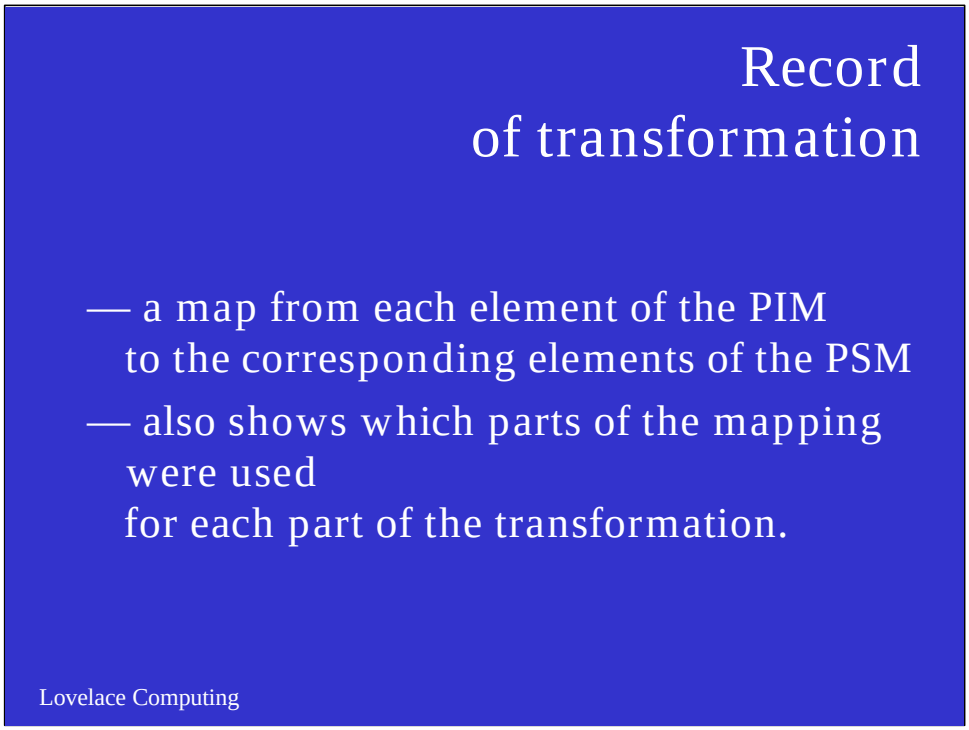
Mark

Transformation

Record

PSM

Lovelace Computing



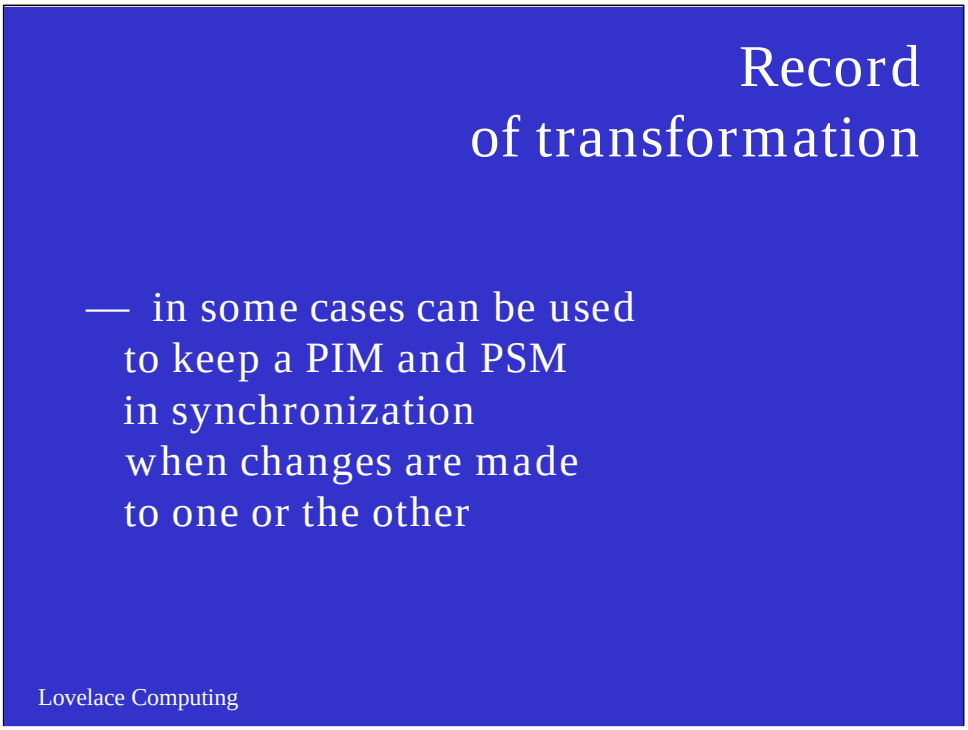
The results of transforming a PIM using a particular technique are a PSM and a record of transformation. The record of transformation includes a map from each element of the PIM to the corresponding elements of the PSM, and shows which parts of the mapping were used for each part of the transformation.

Examples:

A record of transformation shows that a particular class in the PIM becomes three classes in the PSM, related in a certain way.

A record of transformation shows that two objects that were connected directly in the PIM are connected in the PSM via two protocol objects and an intervening interceptor.

The record of transformation can be made available to someone working on either PIM or PSM. An MDA modeling tool that keeps a record of transformation may keep a PIM and PSM in synchronization when changes are made to either.



The results of transforming a PIM using a particular technique are a PSM and a record of transformation. The record of transformation includes a map from each element of the PIM to the corresponding elements of the PSM, and shows which parts of the mapping were used for each part of the transformation.

Examples:

A record of transformation shows that a particular class in the PIM becomes three classes in the PSM, related in a certain way.

A record of transformation shows that two objects that were connected directly in the PIM are connected in the PSM via two protocol objects and an intervening interceptor.

The record of transformation can be made available to someone working on either PIM or PSM. An MDA modeling tool that keeps a record of transformation may keep a PIM and PSM in synchronization when changes are made to either.

How MDA is used

PIM

Mapping

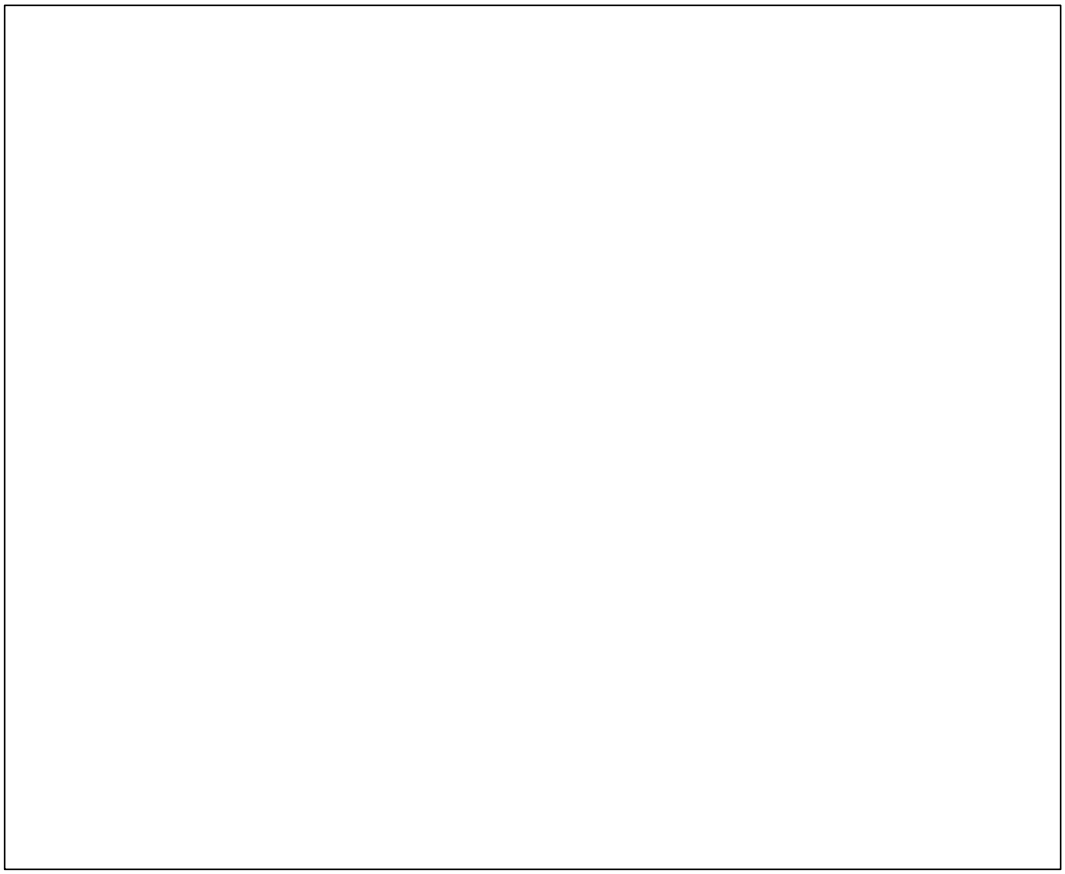
Mark

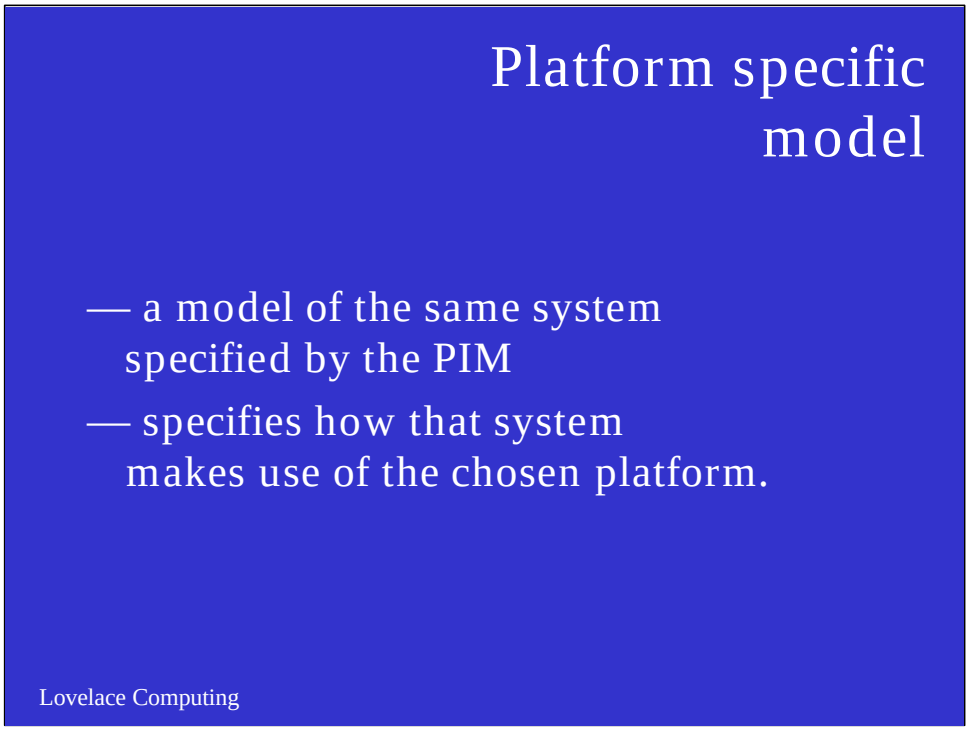
Transformation

Record

PSM

Lovelace Computing





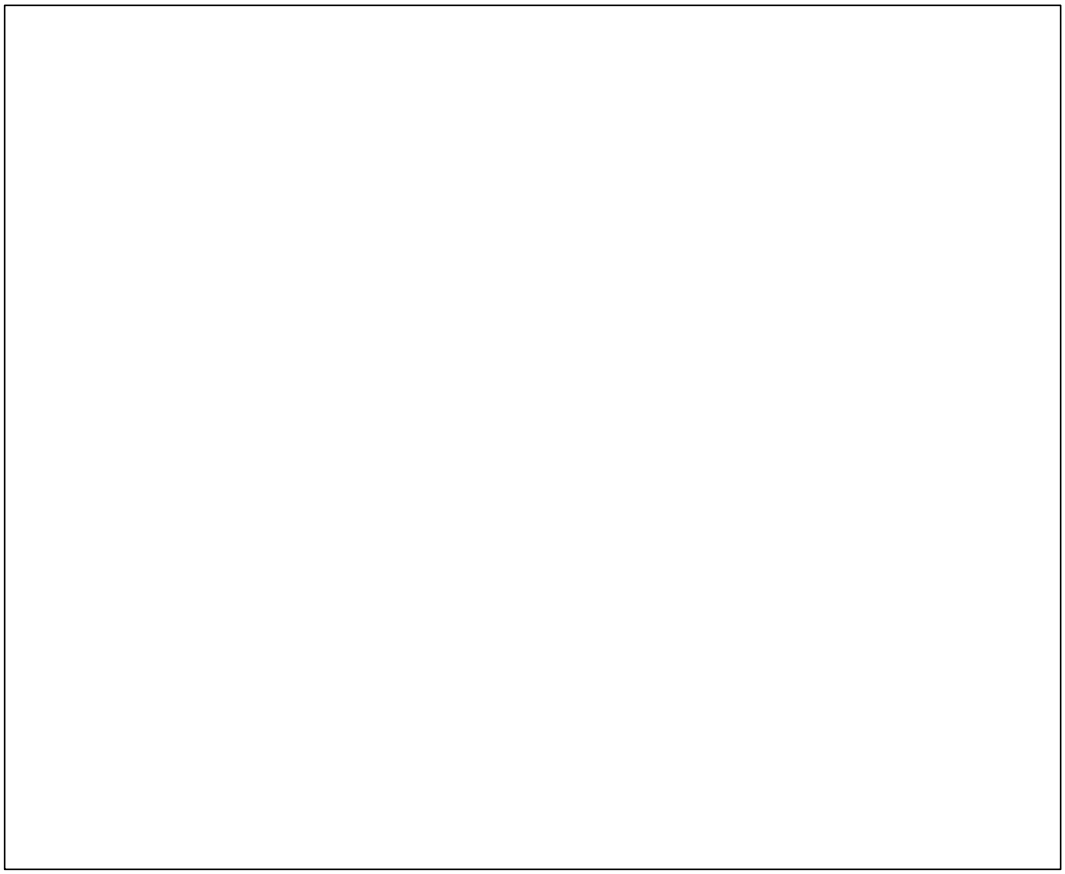
The platform specific model produced by the transformation is a model of the same system specified by the PIM; it also specifies how that system makes use of the chosen platform.

A PSM may provide more or less detail, depending on its purpose. A PSM will be an implementation, if it provides all the information needed to construct a system and to put it into operation.

A PSM that is an implementation will provide a variety of different information, which may include program code, the intended CORBA types of the implementation, program linking and loading specifications, deployment descriptors, and other forms of configuration specifications.

Platform specific model

May provide more or less detail, depending on its purpose.



Implementation

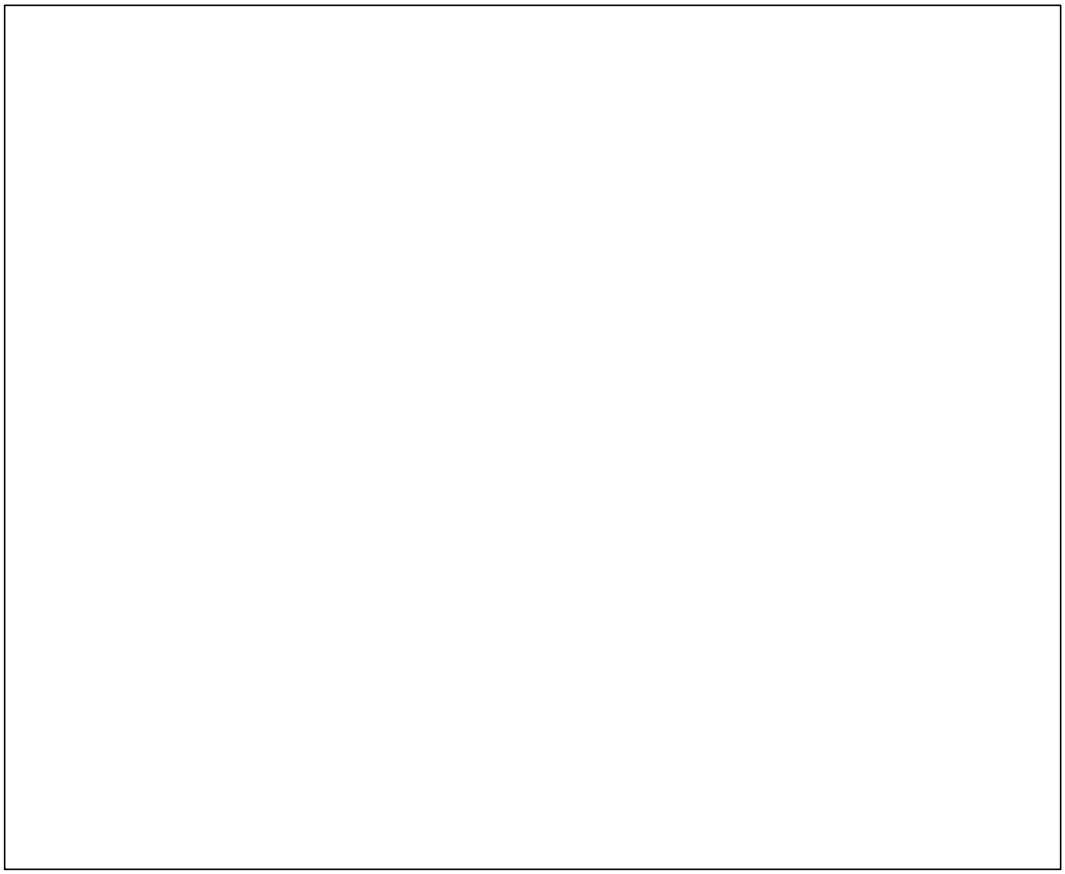
Will be an implementation,
if it provides all the information needed
to construct a system and
to put it into operation.

Lovelace Computing

Implementation

A PSM that is an implementation may include:

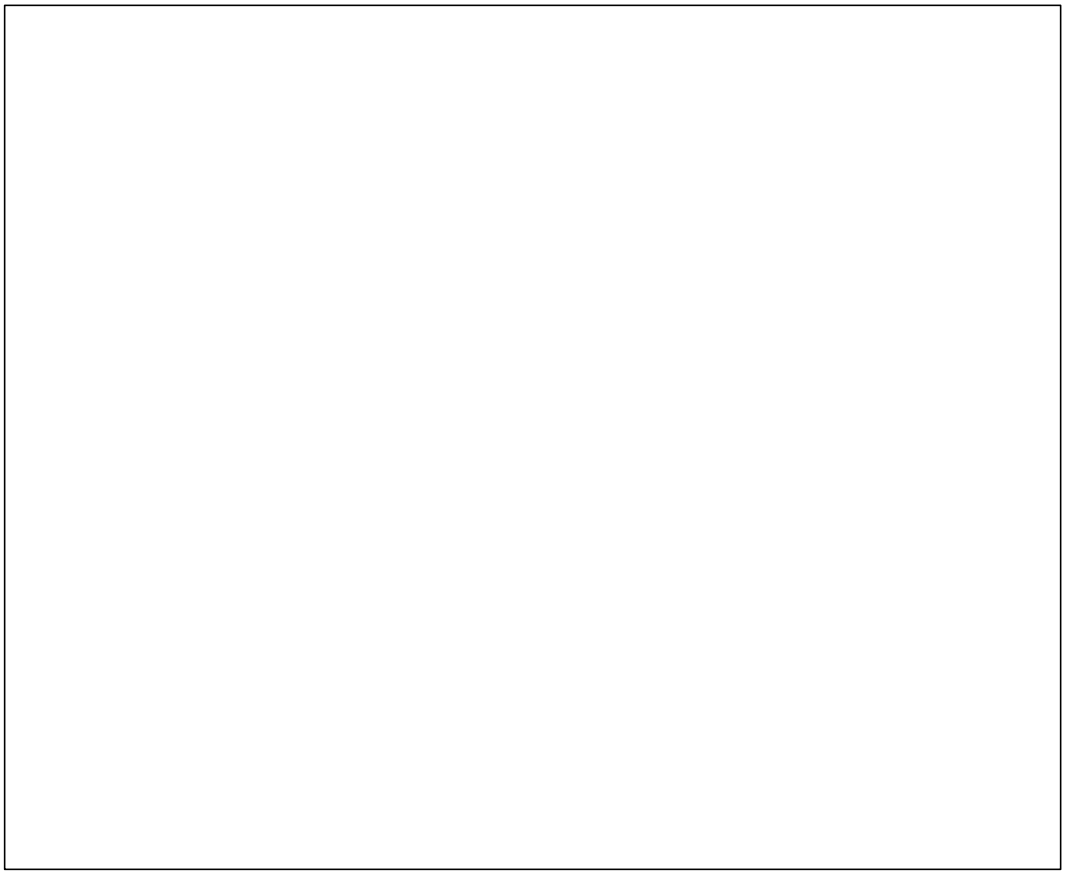
- program code
- intended CORBA types
- linking and loading specifications
- deployment descriptors
- other configuration specifications.



Platform specific model

A PSM that is an implementation may include:

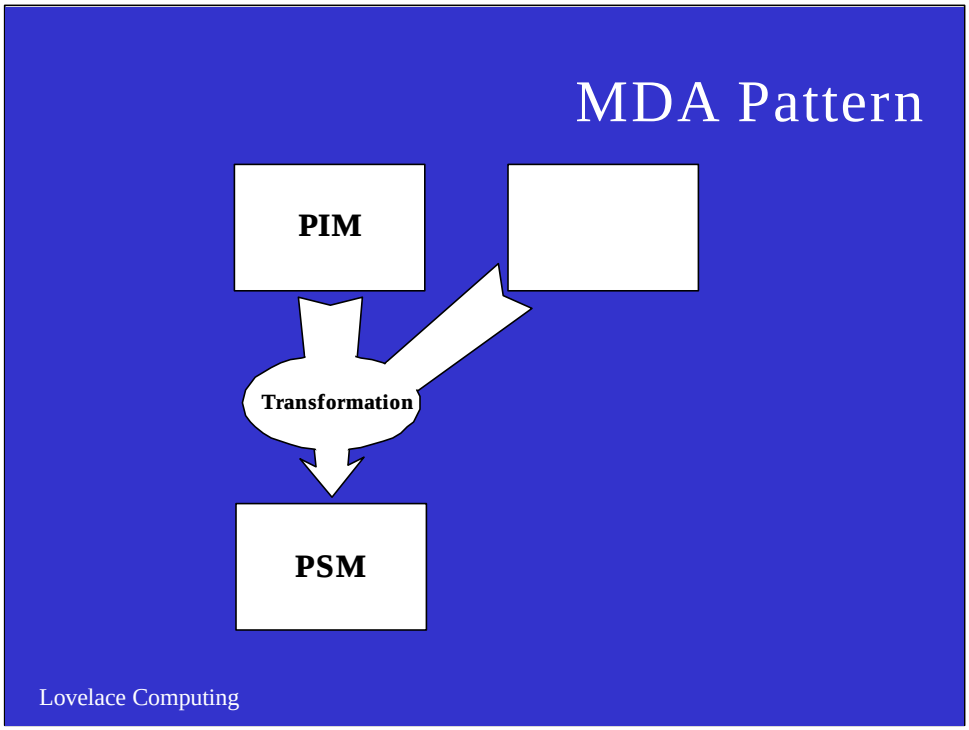
- program code
- intended CORBA or WSDL types
- linking and loading specifications
- deployment descriptors
- other configuration specifications.



How MDA is used

- PIM
- Mapping
- Mark
- Transformation
- Record
- PSM

Lovelace Computing

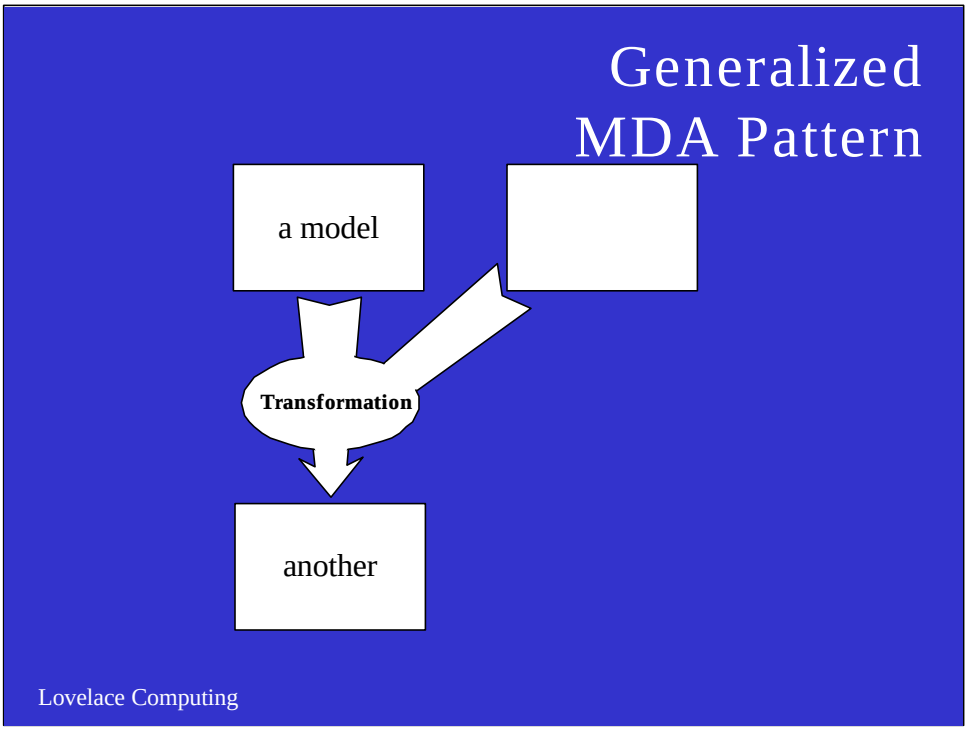


The drawing illustrates the MDA pattern, by which a PIM is transformed to a PSM.

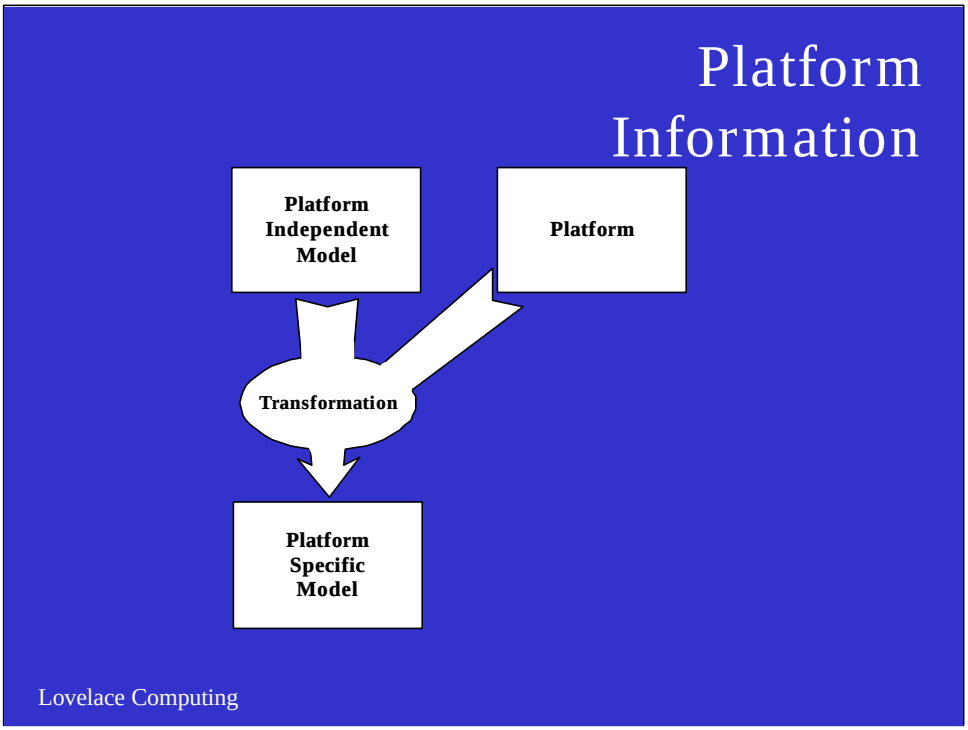
The drawing is intended to be suggestive. The platform independent model and other information are combined by the transformation to produce a platform specific model.

The drawing is also intended to be generic. There are many ways in which such a transformation may be done. However it is done, it produces, from a platform independent model, a model specific to a particular platform.

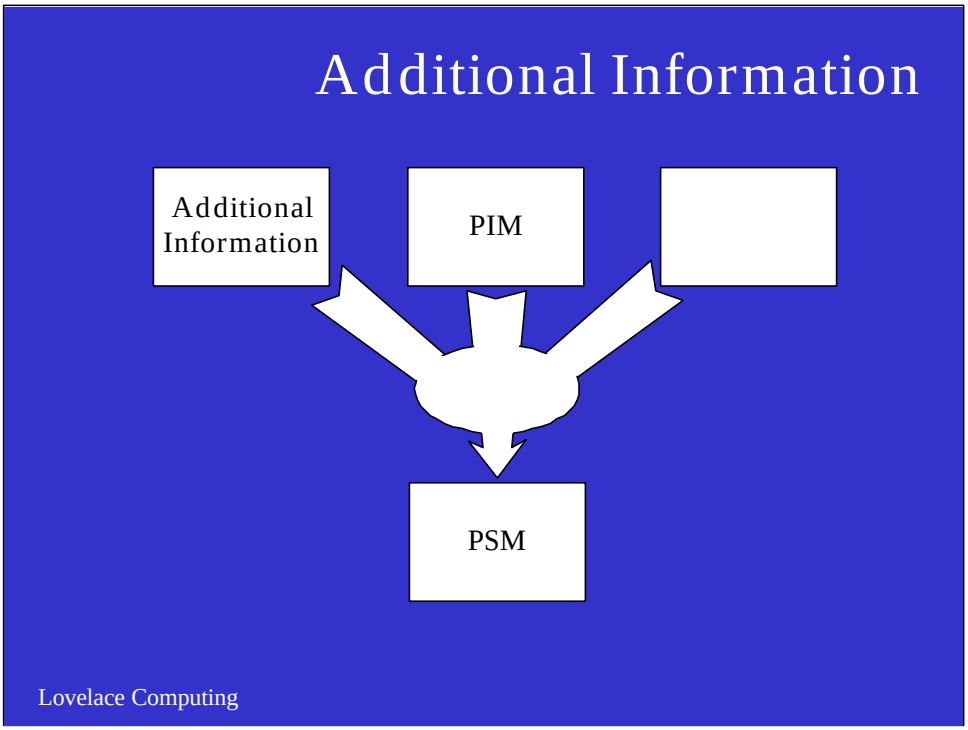
The box with no name represents what goes into the transformation, in addition to the platform independent model. This varies with different styles of MDA.



This drawing is more generic. There are many kinds of model transformations. Many of the same tools that will transform a PIM to a PSM can be used for other kinds of transformation of one kind of model to another.



Information about the chosen platform is required to transform a PIM to a PSM. This information is sometimes imbedded in the transformation tool. Other tools accept information about the platform as input to the transformation process.



The drawing extends the simple MDA pattern to show the use of additional information.

In addition to the PIM and the platform information, additional information can be supplied to guide the transformation.

Examples: A particular architectural style may be specified. Information may be added to connectors to specify quality of service. Selections of particular implementations may be made, where more than one is provided by the transformation. Data access patterns may be specified.

Often the additional information will draw on the practical knowledge of the designer. This will be both knowledge of the application domain and knowledge of the platform.

Transformation approaches

- Marking
- Metamodel transformation
- Model transformation
- Combined Type and Instance
- Pattern application
- Model merging

Transformation approaches

Marking

Metamodel transformation

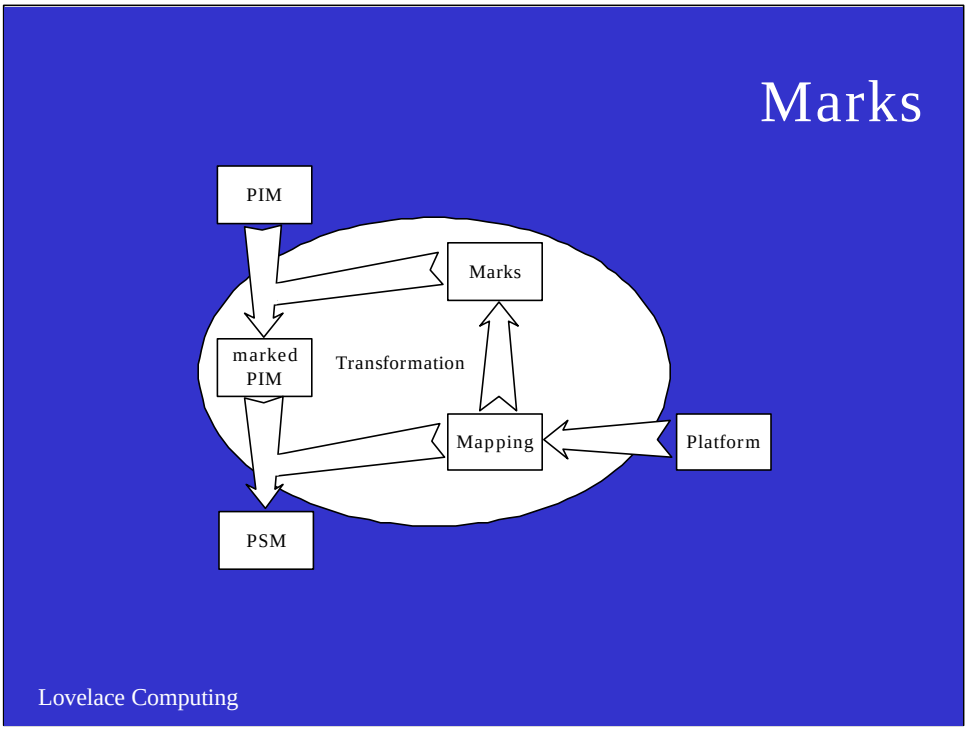
Model transformation

Combined Type and Instance

Pattern application

Model merging

Lovelace Computing



The drawing expands the MDA pattern to show more detail of one of the ways that a transformation may be done.

The drawing is intended to be suggestive. A particular platform is chosen. A mapping for this platform is available or is prepared. This mapping includes a set of marks. The marks are used to mark elements of the model to guide the transformation of the model. The marked PIM is further transformed, using the mapping, to produce the PSM.

Transformation approaches

Marking

Metamodel transformation

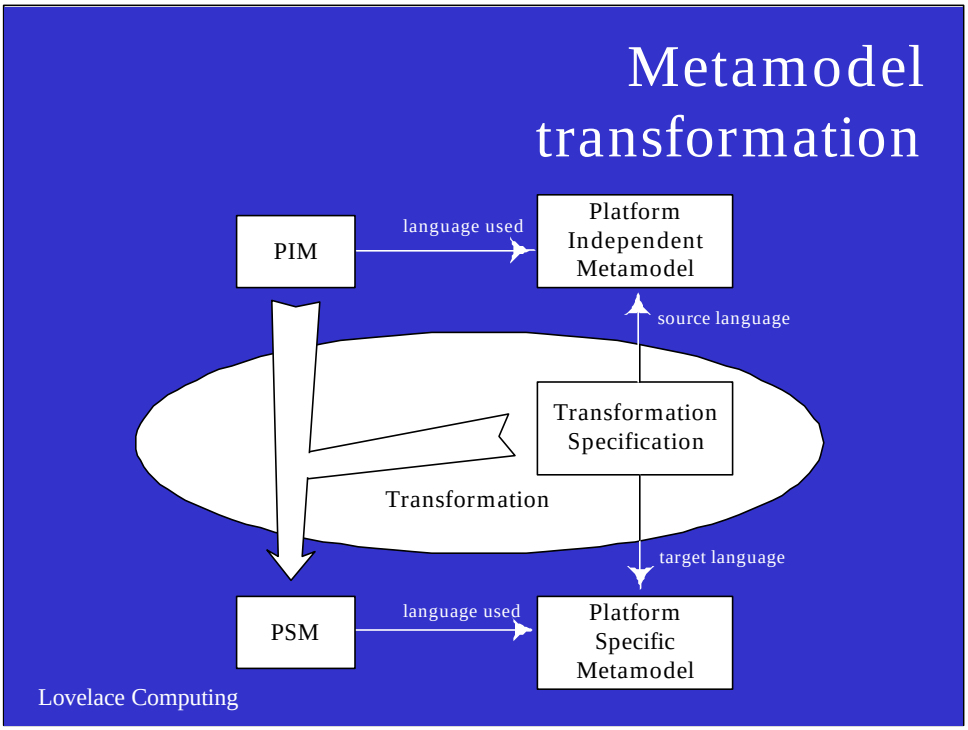
Model transformation

Combined Type and Instance

Pattern application

Model merging

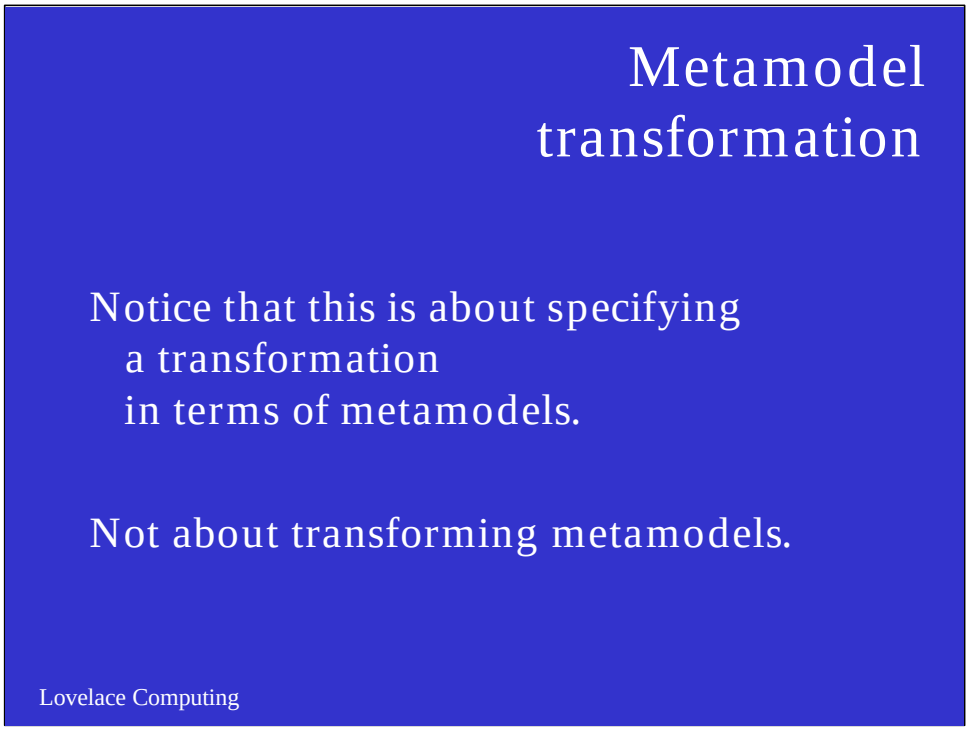
Lovelace Computing



The drawing expands the MDA pattern in a different way, to show more detail of another of the ways that a transformation may be done.

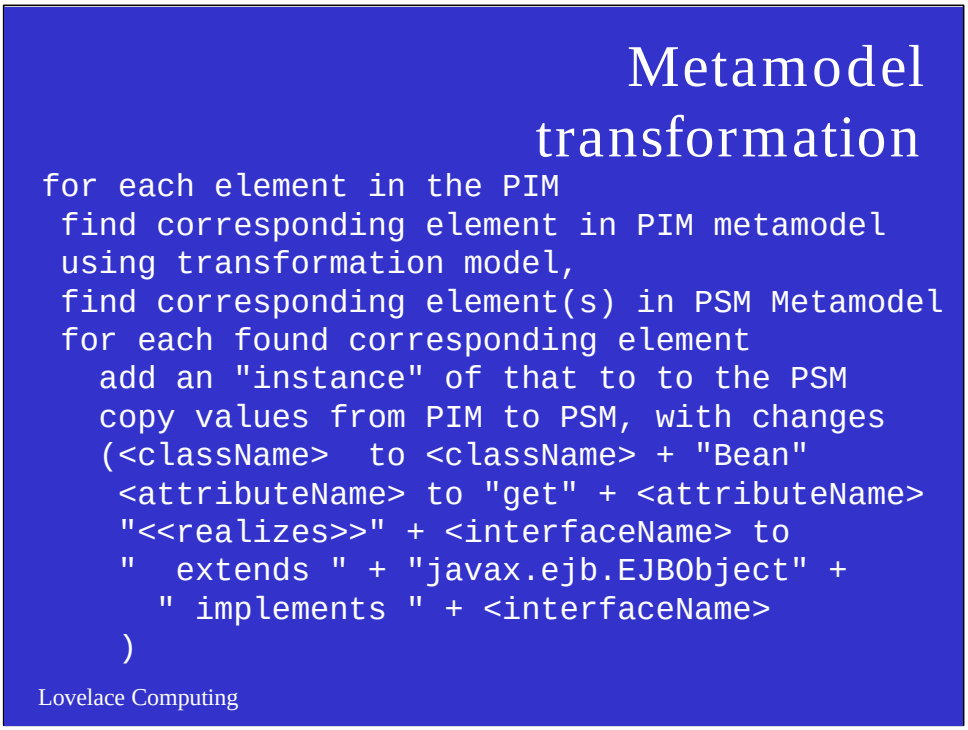
The drawing is intended to be suggestive. A model is prepared using a platform independent language specified by a metamodel. A particular platform is chosen. A specification of a transformation for this platform is available or is prepared. This transformation specification is in terms of a mapping between metamodels. The mapping guides the transformation of the PIM to produce the PSM.

Example: The platform independent metamodel is the EDOC ECA Business Process Model, and the platform specific metamodel is a MOF model of a workflow engine. The transformation specification is a MOF QVT transformation model. The transformation is carried out by a transformation engine created by a tool, which uses a pair of MOF models to build an engine for a specific transformation.



Notice that Metamodel transformation is about specifying a transformation in terms of metamodels.

It is not about transforming metamodels. Of course, metamodels are models. So one metamodel can be transformed into another, using the same general model transformation techniques as are used with any other models.



Here is a simplified transformation specification, using the metamodel transformation style.

```
for each element in the PIM
 find corresponding element in PIM metamodel
 using transformation model,
 find corresponding element(s) in PSM Metamodel
 for each found corresponding element
   add an "instance" of that to to the PSM
   copy values from PIM to PSM, with changes
   (<className>  to <className> + "Bean"
    <attributeName> to "get" + <attributeName>
    "<<realizes>>" + <interfaceName> to
      "  extends " + "javax.ejb.EJBObject" +
      " implements " + <interfaceName>
    )
```

Metamodel transformation

```
for each element in the PIM
 find corresponding element in PIM metamodel
 using transformation model,
 find corresponding element(s) in PSM Metamodel
 for each found corresponding element
   add an "instance" of that to to the PSM
   copy values from PIM to PSM, with changes
   (<className>  to <className> + "Bean"
    <attributeName> to "get" + <attributeName>
    "<<realizes>>" + <interfaceName> to
    "  extends " + "javax.ejb.EJBObject" +
      " implements " + <interfaceName>
    )
```

Lovelace Computing

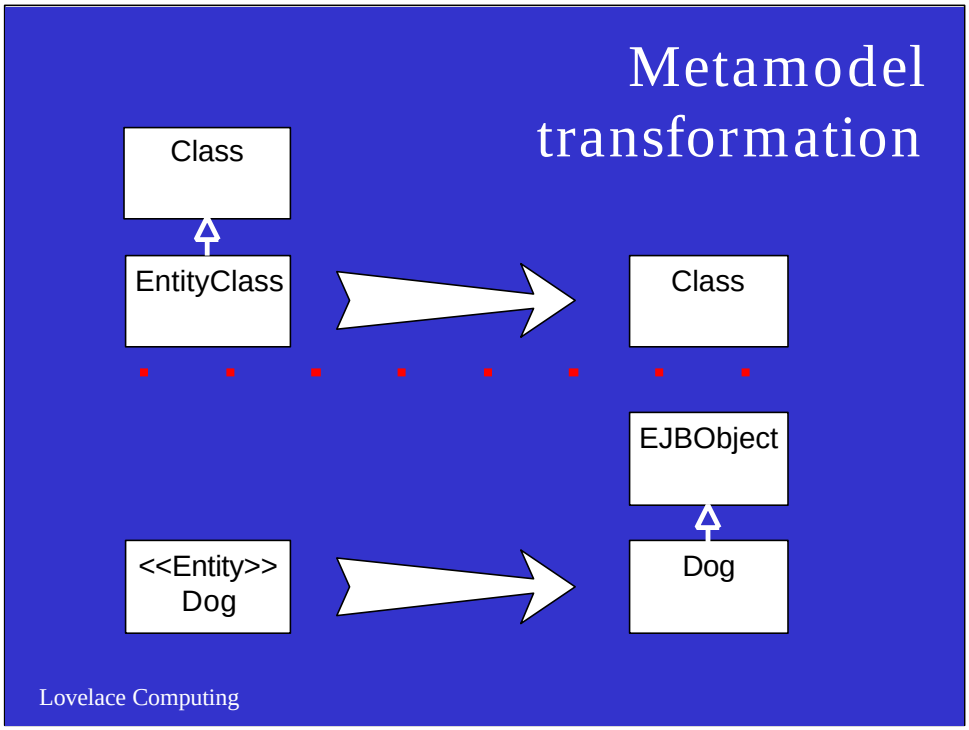
Notice:

```
"<<realizes>>" + <interfaceName> to
  "  extends " + "javax.ejb.EJBObject" +
  " implements " + <interfaceName>
```

This mentions the class, `javax.ejb.EJBObject`. This class does not appear in the target metamodel. It is a class in the PSM.

It is clear to me that there is something wrong with the explanation of this style in the MDA Guide. Even though practitioners of this style were active in preparation of the Guide.

(As I wrote earlier, I'm to blame, not only for errors in this tutorial, but, as editor, for errors in the MDA Guide.)



The class EJBObject does not, can not, appear in the target metamodel. So, in this case, anyway, the transformation can be specified in the language of the metamodels.

Metamodel transformation

```
for each element in the PIM
 find corresponding element in PIM metamodel
 using transformation model,
 find corresponding element(s) in PSM Metamodel
 for each found corresponding element
   add an "instance" of that to to the PSM
   copy values from PIM to PSM, with changes
   (<className>  to <className> + "Bean"
    <attributeName> to "get" + <attributeName>
    "<<realizes>>" + <interfaceName> to
    "  extends " + "javax.ejb.EJBObject" +
      " implements " + <interfaceName>
    )
```

Lovelace Computing

Actually, we have skipped past a simpler example. “Bean” is a string. There are no strings in the target metamodel. There is, of course, the class, String, but that is a class, not a string. “Bean” can’t appear in the target metamodel.

Transformation approaches

Marking

Metamodel transformation

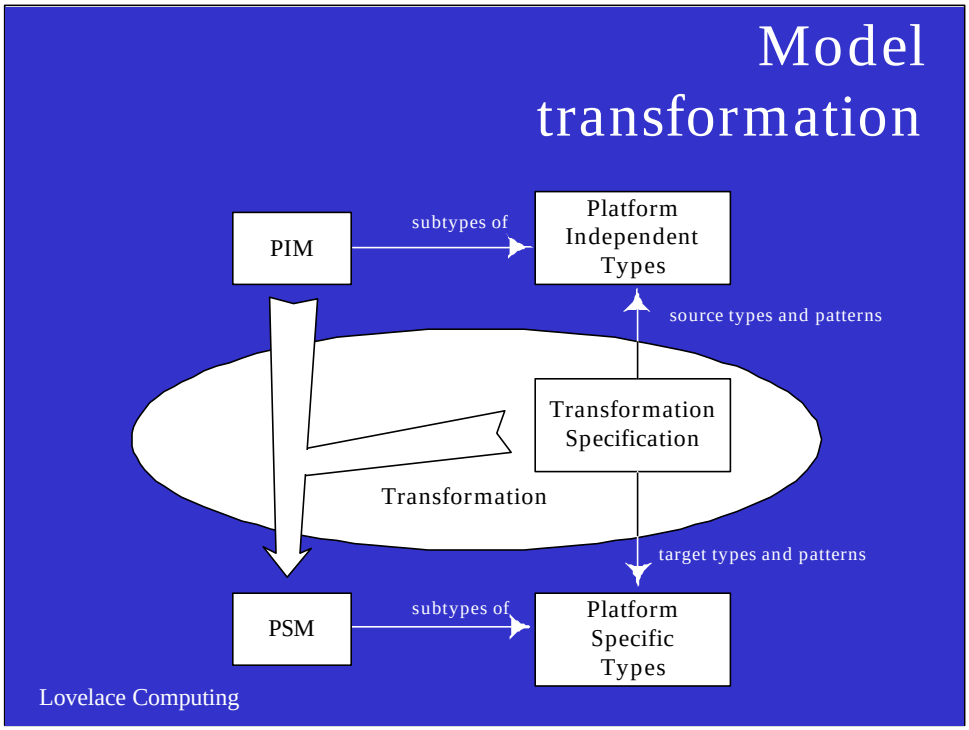
Model transformation

Combined Type and Instance

Pattern application

Model merging

Lovelace Computing

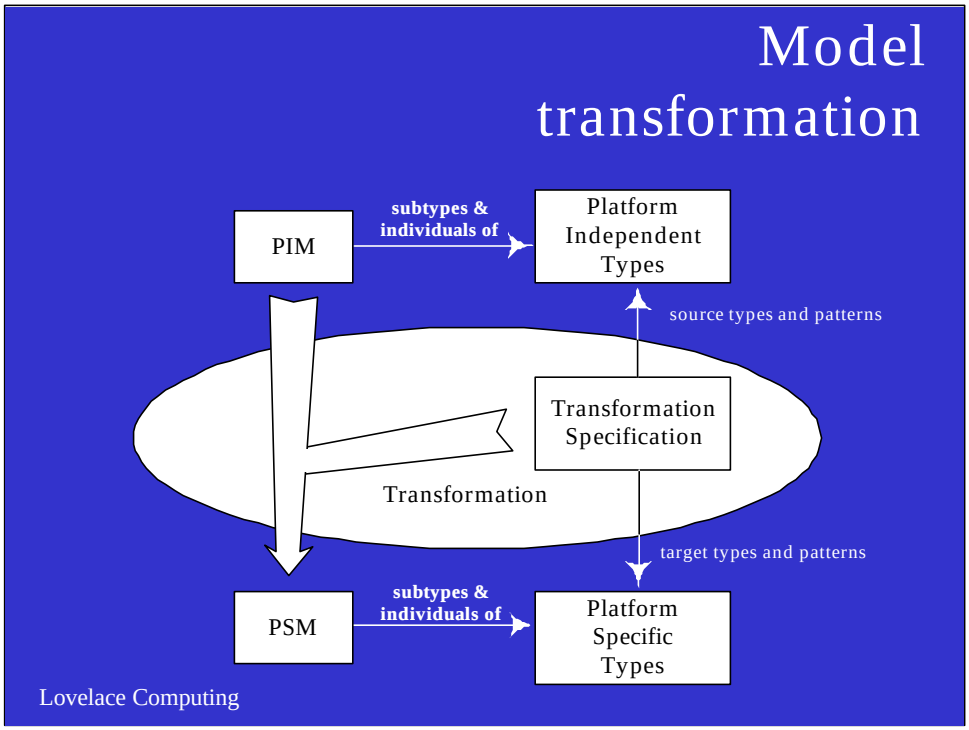


The drawing shows yet another of the ways that a transformation may be done.

The drawing is intended to be suggestive. A model is prepared using platform independent types specified in a model. The types may be part of a software framework. The elements in the PIM are subtypes of the platform independent types. A particular platform is chosen. A specification of a transformation for this platform is available or is prepared. This transformation specification is in terms of a mapping between the platform independent types and the platform dependent types. The elements in the PSM are subtypes of the platform specific types.

Example: The platform independent types declare generic capabilities and features. The platform specific types are mix-in classes and composite classes that provide the capabilities and features specific to a particular type of platform.

This approach differs from metamodel mapping primarily in that types specified in a model are used for the mapping, instead of concepts specified by a metamodel.



Here again, we will often need to use specific individuals, in addition to types. So the text on the horizontal arrows needs to be changed to read ’subtypes and individuals of.’

Transformation approaches

- Marking
- Metamodel transformation
- Model transformation
- **Combined Type and Instance**
- Pattern application
- Model merging

Combined type and instance mapping

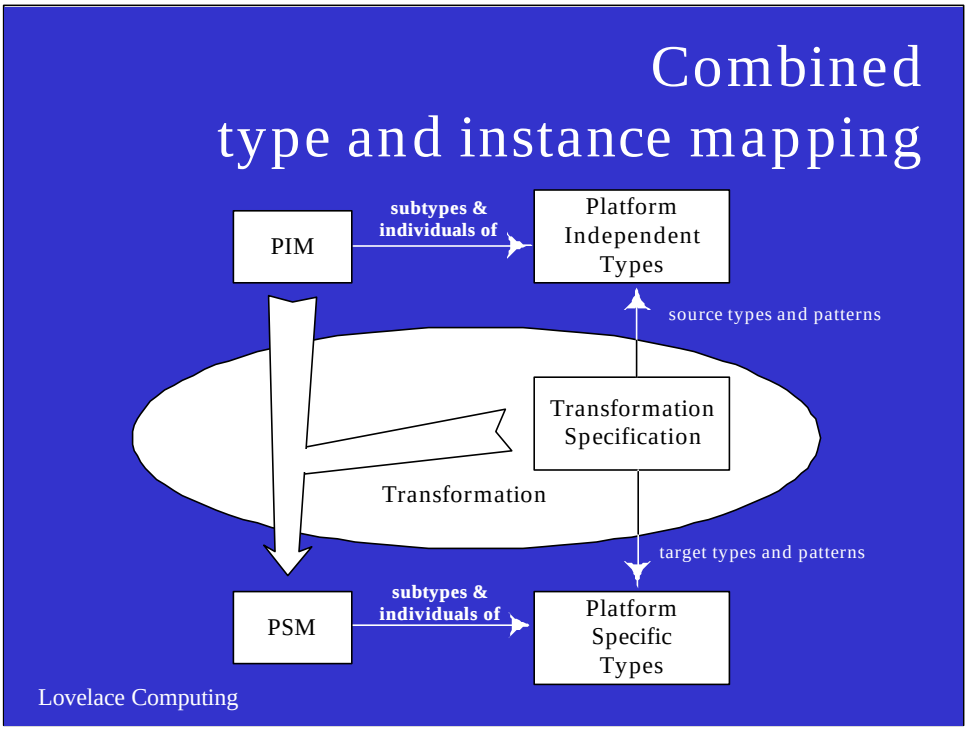
Combines type and instance mapping.

Lovelace Computing

Most mappings, will necessarily consist of some combination of the above approaches.

A model type mapping is only capable of expressing transformations in terms of rules about things of one type in the PIM resulting in the generation of some thing(s) of some (one or more) type(s) in the PSM. However, without the ability for the architect to also mark the model with additional information for use by the transformation, the mapping will be deterministic, and will rely wholly on Platform Independent information to generate the PSM. Rules in the mapping will often specify that certain types in the PIM must be marked with one of a set of marks in order that the PSM will have the right non-functional or stylistic characteristics, which cannot be determined from information in the PIM.

Likewise, every transformation of model instances has implicit type constraints which the architect marking the model must obey in order for the transformation to make sense. For example, marking an Association End in a UML model with the mark, 'Entity,' makes no sense, whereas marking it with the mark, 'RMI navigable,' does. Implicitly each type of model element in the PIM is only suitable for certain marks, which indicate what type of model element will be generated in the PSM. Transformations based on marking instances will either explicitly state which marks are suitable for which types in the PIM, or these type constraints will be implicitly understood by the user of the marks.

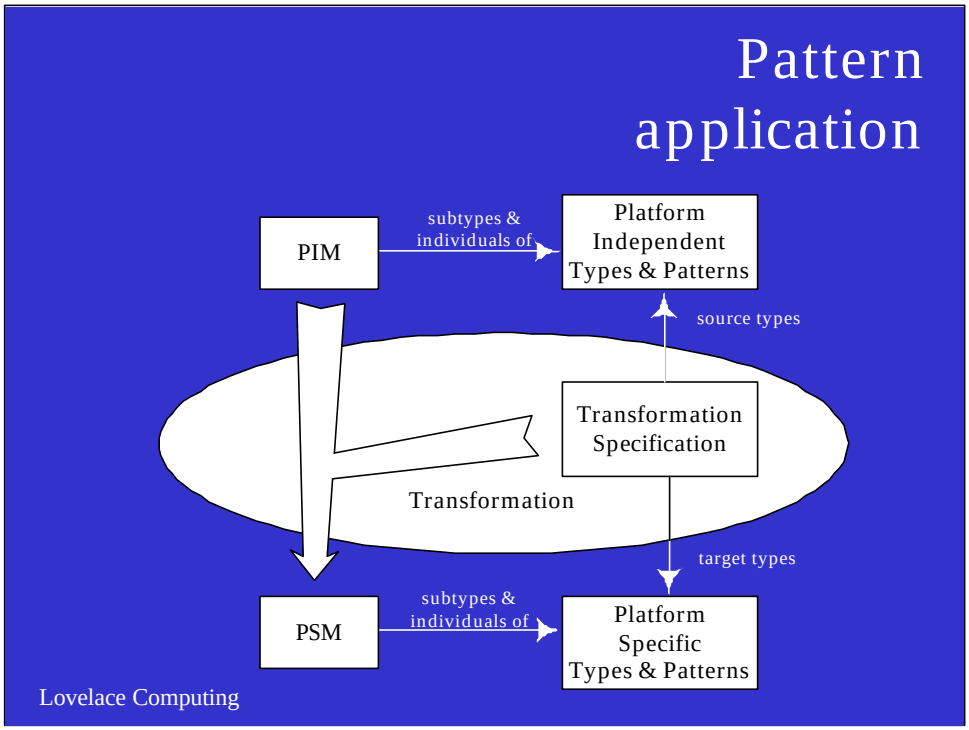


You see that this drawing is identical to the previous drawing.

Transformation approaches

- Marking
- Metamodel transformation
- Model transformation
- Combined Type and Instance
- **Pattern application**
- Model merging

Lovelace Computing

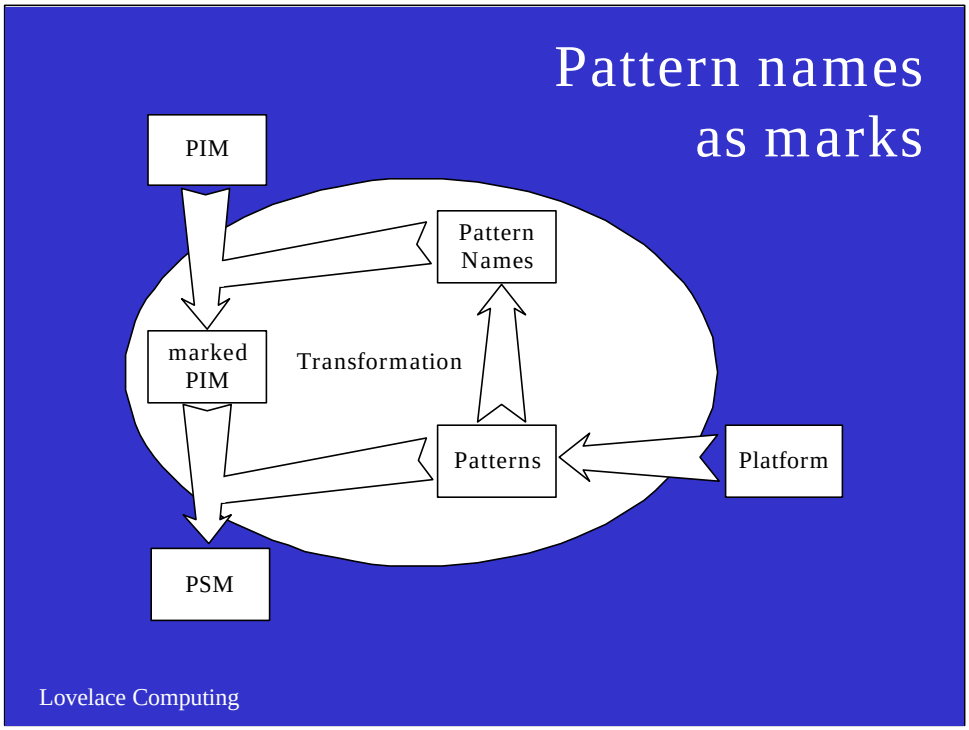


Extension of the model and metamodel mapping approaches include patterns along with the types or the modeling language concepts.

In addition to platform independent types, a generic model can supply patterns. Both the types and patterns can be mapped to platform specific types and patterns.

Example: A platform independent model uses a generic model defining object types corresponding to the concepts of the RM-ODP Engineering Language, and patterns for their use, corresponding to the structuring rules of the Engineering Language. The transformation specification maps these types to object types to be used in a CORBA implementation, and these patterns to corresponding patterns in the Common ORB Architecture. ODP stubs become CORBA stubs and skeletons; the functions of ODP binders are mapped to ORB and object adapter functions; ODP interceptors become CORBA interceptors…

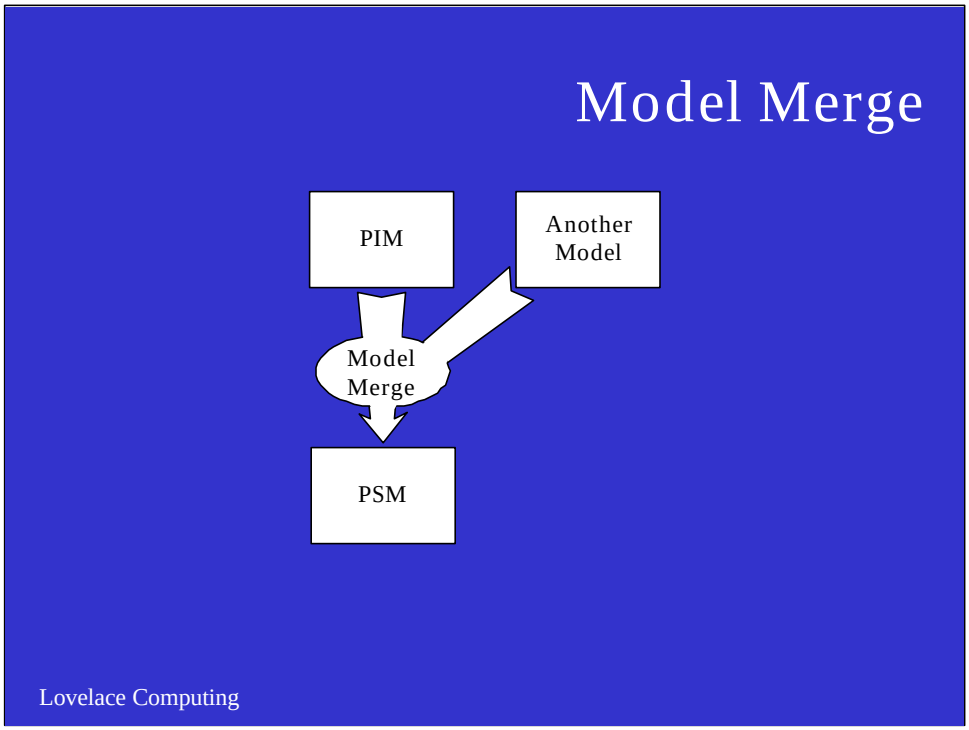
The metamodel mapping approach can use patterns in the same way.



The drawing shows another way to use patterns: as the names of platform specific marks, that is, the names of design patterns that are specific to a platform.

Transformation approaches

- Marking
- Metamodel transformation
- Model transformation
- Combined Type and Instance
- Pattern application
- **Model merging**



The drawing expands the MDA pattern in a different way to show more detail of another one of the ways that a transformation may be done.

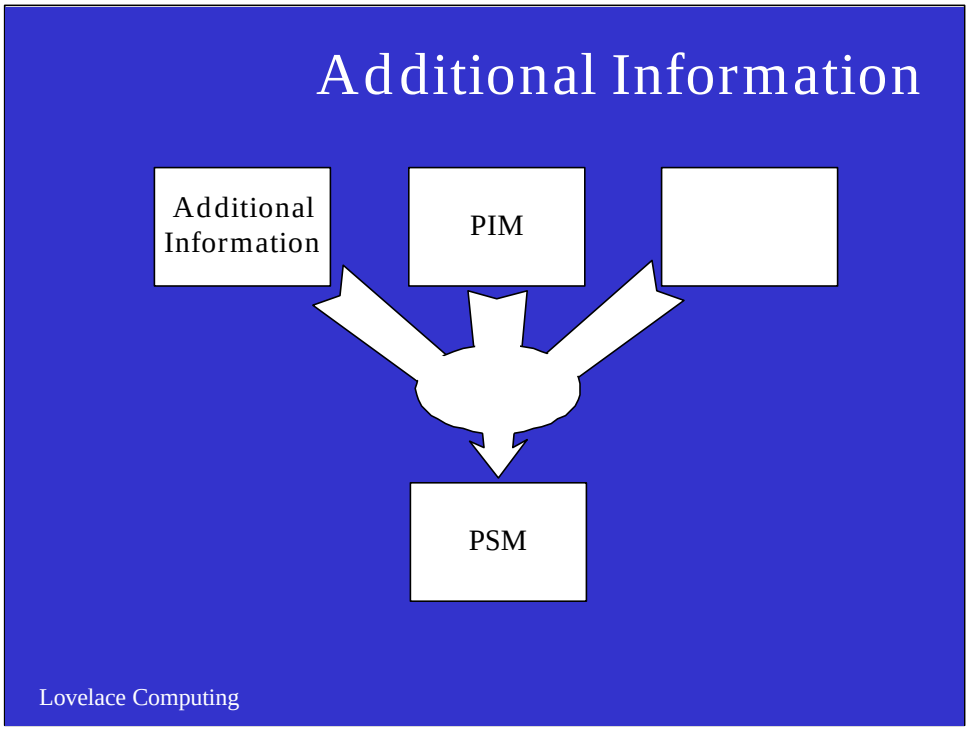
Again, the drawing is intended to be suggestive. It is also generic. There are several MDA approaches that are based on merging models.

An earlier example shows the use of patterns and pattern application. At least some cases of pattern application can be regarded as one kind of model merging.

The 2U submission for UML 2 proposed model merging as a technique for language specification. Much of the proposed technique has been incorporated in the adopted UML 2.

Transformation approaches

- Marking
- Metamodel transformation
- Model transformation
- Combined Type and Instance
- Pattern application
- Model merging

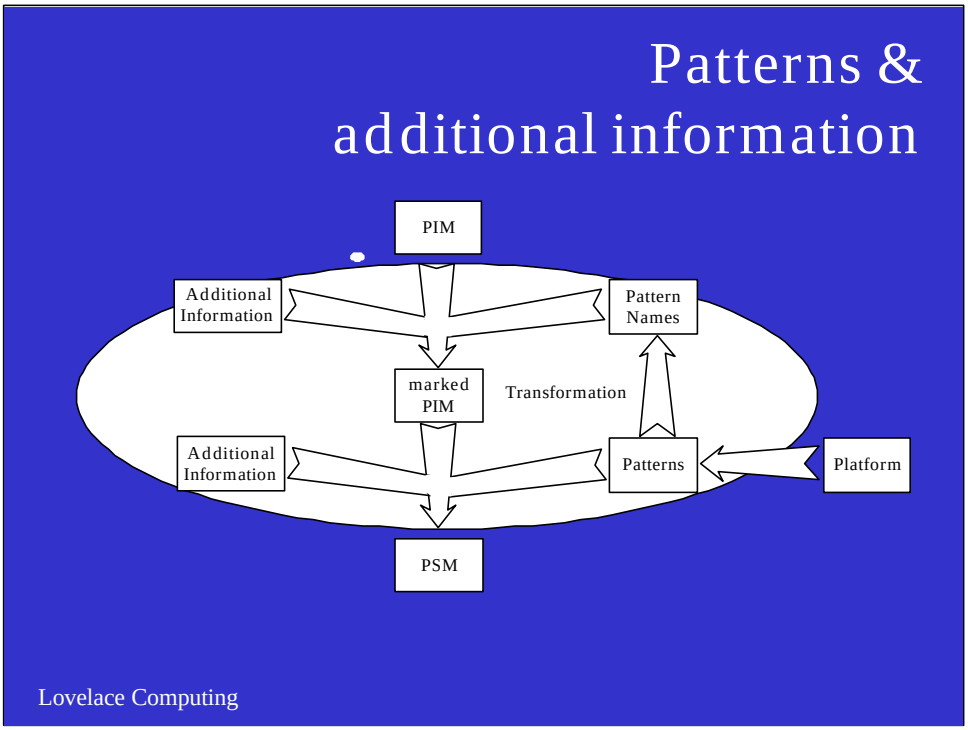


The drawing extends the simple MDA pattern to show the use of additional information.

In addition to the PIM and the platform specific marks, additional information can be supplied to guide the transformation.

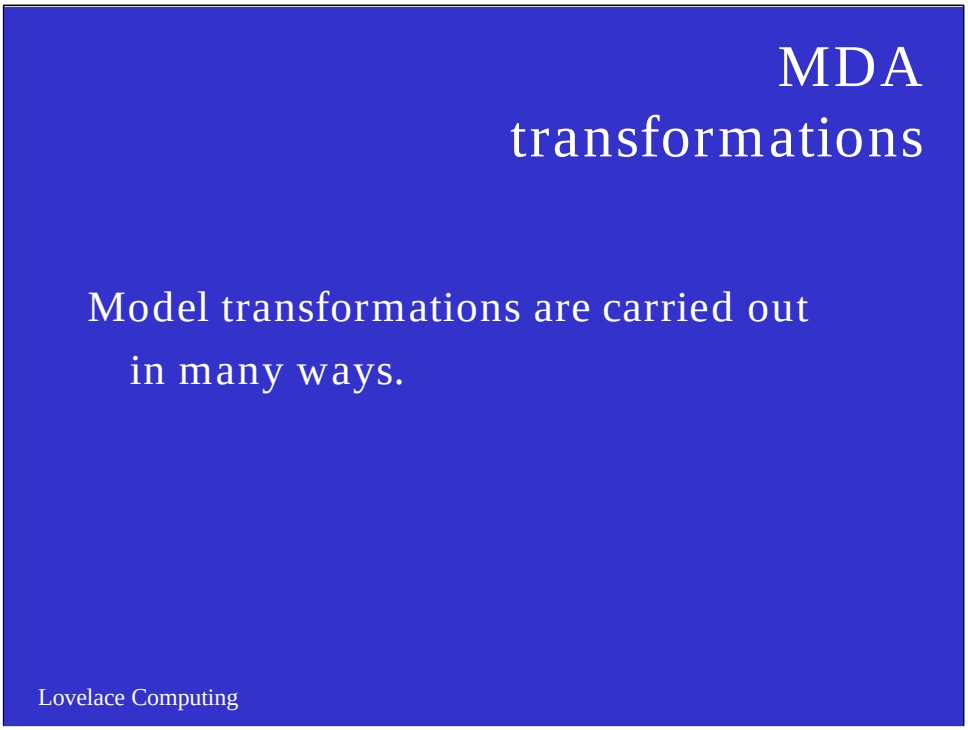
Examples: A particular architectural style may be specified. Information may be added to connectors to specify quality of service. Selections of particular implementations may be made, where more than one is provided by the transformation. Data access patterns may be specified.

Often the additional information will draw on the practical knowledge of the designer. This will be both knowledge of the application domain and knowledge of the platform.



The drawing further expands the MDA pattern to show the use of additional information in a particular transformation technique.

The drawing is intended to be suggestive. In the process of preparing a PIM, in addition to using the pattern names provided, other information can be added to produce the marked PIM. More information, in addition to the patterns, can be used when the marked PIM is further transformed to produce the PSM.



There is a range of tool support for model transformation. Transformations can use different mixtures of manual and automatic transformation. There are different approaches to putting into a model the information necessary for a transformation from PIM to PSM. Four different transformation approaches described here illustrate the range of possibilities: manual transformation, transforming a PIM that is prepared using a profile, transformation using patterns and markings, and automatic transformation.

MDA transformations

- Manual
- Using a profile
- Using patterns and markings
- Automatic

In order to make the transformation from PIM to PSM, design decisions must be made. These design decisions can be made during the process of developing a design that conforms to engineering requirements on the implementation. This is a useful approach, because these decisions are considered and taken in the context of a specific implementation design.

This manual transformation process is not greatly different from how much good software design work has been done for years. The MDA approach adds value in two ways:

— the explicit distinction between a platform independent model and the transformed platform specific model,

— the record of the transformation.

MDA transformations

Manual

Using a profile

Using patterns and markings

Automatic

Lovelace Computing

A PIM may be prepared using a platform independent UML profile. This model may be transformed into a PSM expressed using a second, platform specific UML profile.

The transformation may involve marking the PIM using marks provided with the platform specific profile.

A PIM may be prepared using a platform independent UML profile. This model may be transformed into a PSM expressed using a second, platform specific UML profile.

The transformation may involve marking the PIM using marks provided with the platform specific profile.

It can be argued that this distinction is spurious: What is the difference between using a profile as a language for a PIM and/or a PSM, and using plain old UML, or a metamodel? As we know there are some models expressible as either a UML profile, or in some other language (e.g. CORBA – UML Profile, IDL, or CCM metamodel, ECA – UML Profile or metamodel). This does not dictate the way in which transformations can be achieved. In fact UML Profiles are quite amenable to metamodel transformation, as UML has a valid MOF metamodel – in UML 2.0 this will be even more direct.

MDA transformations

Manual

Using a profile

Using patterns and markings

Automatic

Lovelace Computing

Patterns may be used in the specification of a mapping. The mapping includes a pattern and marks corresponding to some elements of that pattern.

In model instance transformations the specified marks are then used to prepare a marked PIM. The marked elements of the PIM are transformed according to the pattern to produce the PSM.

Example: A decorator pattern with two roles, decoration and decorated supplied a mark, decorated. When this mark is applied to a class in a model, the transformation might produce a class corresponding to that class, with additional operations and attribute, a new class, corresponding to the decoration, and an association between those classes.

MDA transformations

Manual

Using a profile

Using patterns and markings

Automatic

Lovelace Computing

Several patterns may be combined to produce a new pattern. New marks can then be specified for use with the new pattern.

In model type transformations rules will specify that all elements in the PIM which match a particular pattern will be transformed into instances of another pattern in the PSM. The marks will be used to bind values in the matched part of the PIM to the appropriate slots in the generated PSM. In this usage the target patterns can be thought of as templates for generating the PSM, and the use of marks as a way of binding the template parameters.

Example: A mapping from EDOC ECA to EJB might include a pattern of ECA types identifying appropriate ProcessComponents and their associated document types as suitable for mapping to EJB Entities and their Remote Interfaces and container managed data classes. Marks in the source pattern will correspond to marks in the target pattern. For example a mark, 'Name,' might be used to identify the attribute, 'name,' of each matched ProcessComponent and make it the classname of the Entity's Remote Interface.

There are contexts in which a PIM can provide all the information needed for implementation, and there is no need to add marks or use data from additional profiles, in order to be able to generate code. One such is that of mature component-based development, where middleware provides a full set of services, and where the necessary architectural decisions are made once for a number of projects, all building similar systems (for example, there is a component based product line architecture in place). These decisions are implemented in tools, development processes, templates, program libraries, and code generators.

In such a context, it is possible for an application developer to build a PIM that is complete as to classification, structure, invariants, and pre- and post-conditions. The developer can then specify the required behavior directly in the model, using an action language. This makes the PIM computationally complete; that is, the PIM contains all the information necessary to produce computer program code.

In this context, the developer need never see a PSM, nor is it necessary to add additional information to the PIM, other than that already available to the transformation tool. The tool interprets the model directly or transforms the model directly to program code.

Such a PIM, in a mature component development shop, with an established architectural style and with platform specific engineering decisions already made and being reused, can be used to generate code (i.e. components in their code form) not only to different CORBA Components or J2EE platforms, but also to some of the other application server platforms.

This assumes that someone has prepared for re-use:

(a) a model of the architectural style
(b) detail within that model, such as a PIM type system, that can be automatically mapped to the various target platforms
(c) the necessary tool support to deliver the model to the developers in the form of profiles, model conformance checks, links to an IDE, supporting processes, and so forth
(d) a mapping for each target platform.

The point is that, with such development environment support, for a given application, the application developer need develop only a PIM, and code can be directly generated from that PIM.

The information that would otherwise be in a visible PSM is effectively pre-packaged, and provided to the application developer within the development environment.

As mentioned, there may be an advantage to providing the developer a model of the generated code.

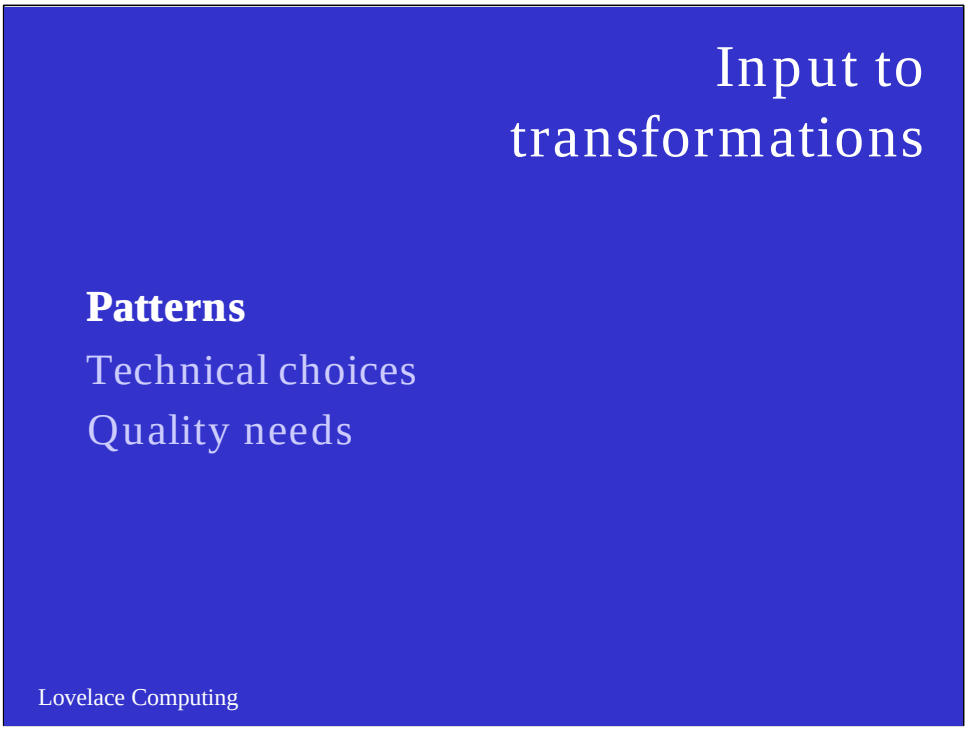
MDA transformations

- Manual
- Using a profile
- Using patterns and markings
- Automatic

Input to transformations

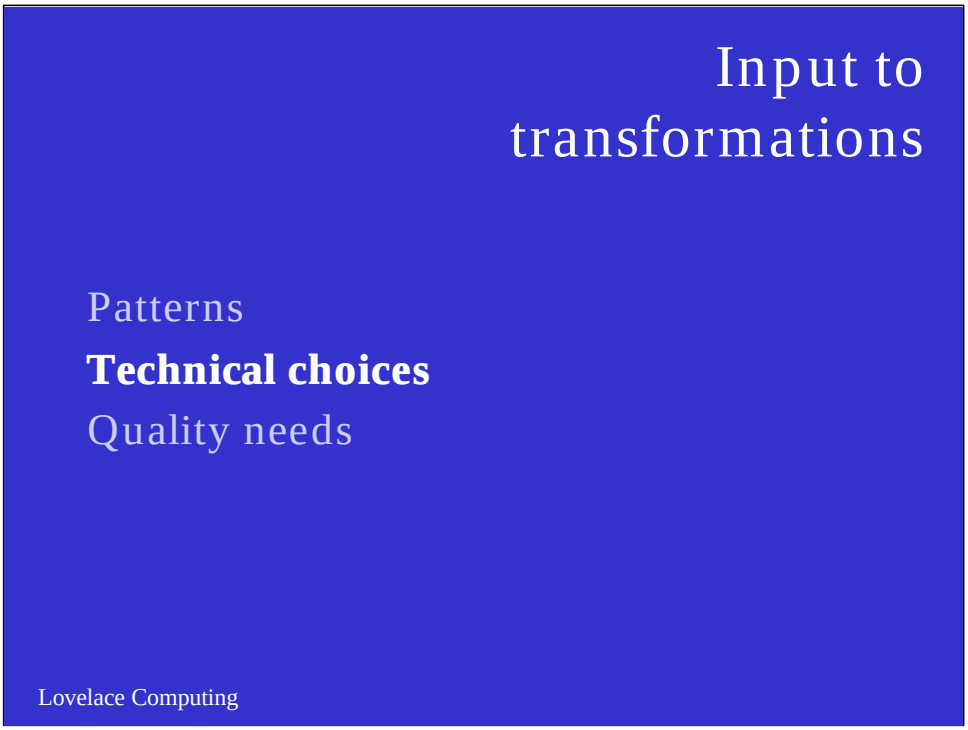
- Patterns
- Technical choices
- Quality needs

Lovelace Computing

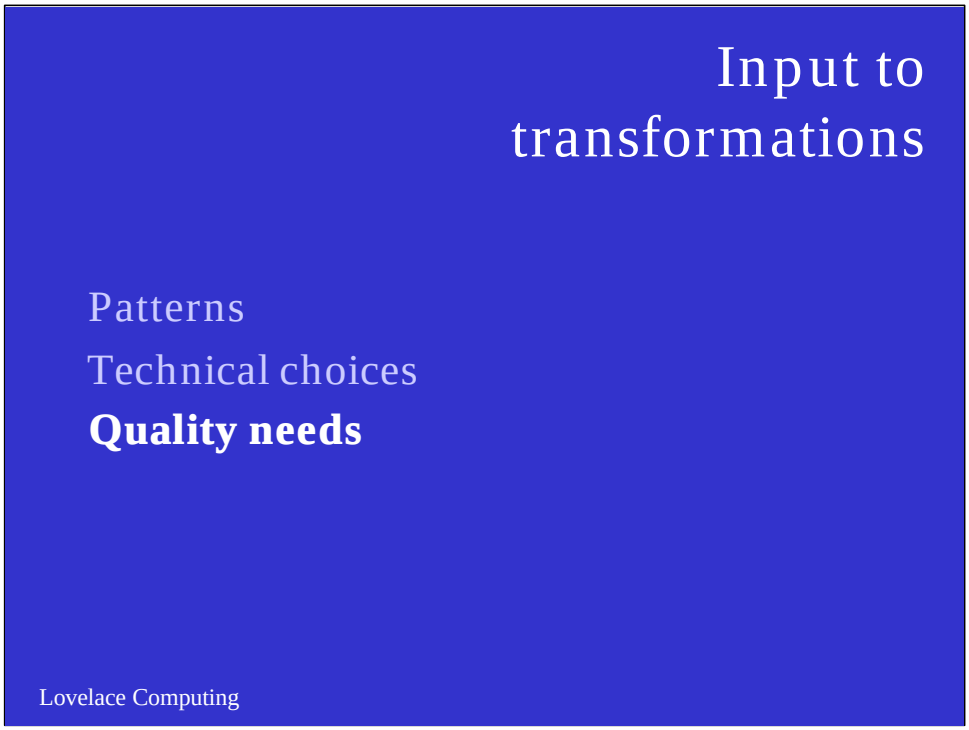


Generic transformation techniques can work with patterns supplied by the architect or builder. Different patterns, chosen by the architect, or by a transformation tool using supplied selection criteria.

Patterns are also important in the description of groups of concepts in one model that correspond to a concept, or different group of concepts in another model when specifying a type-based transformation. Tools will then be responsible for matching the patterns in the source model and using the patterns in the target model as templates for creating the new model.



Technical choices of all kinds can be made by the architect or builder and used to guide the transformation. Technical choices might also be made by analysis tools working with the PIM, and then used in manual or automatic transformation. Most approaches will use some combination of some automated transformation with architect-chosen manual input to the transformation.



A whole range of quality of service requirements can be used to guide transformations. In a transformation to a PIM, specific transformation choices will be made according to the particular qualities required at each conformance point in the model.

Input to transformations

- Patterns
- Technical choices
- Quality needs

Other MDA capabilities

- Multi-platform models
- Federated systems
- Multiple transformations
- General transformations
- Reuse of mappings

Lovelace Computing

So far, we have focused on a straightforward PIM to PSM transformation. Now, let's discuss several other uses of Model-Driven Architecture.

Other MDA capabilities

Multi-platform models

Federated systems

Multiple transformations

General transformations

Reuse of mappings

Lovelace Computing

Many systems are built on more than one platform. An MDA transformation can use marks from several different platform models to transform a PIM into a PSM with parts of the system on several different platforms.

Example: A trading system PIM is transformed to a web services front end and a mainframe back office system.

Example: A system needs to communicate with several existing systems. Several means of communication are available, IIOP, RMI, and SOAP. The architect chooses the means most suitable for each connector and marks that connector with a mark from the set for that means.

Other MDA capabilities

Multi-platform models

Federated systems

Multiple transformations

General transformations

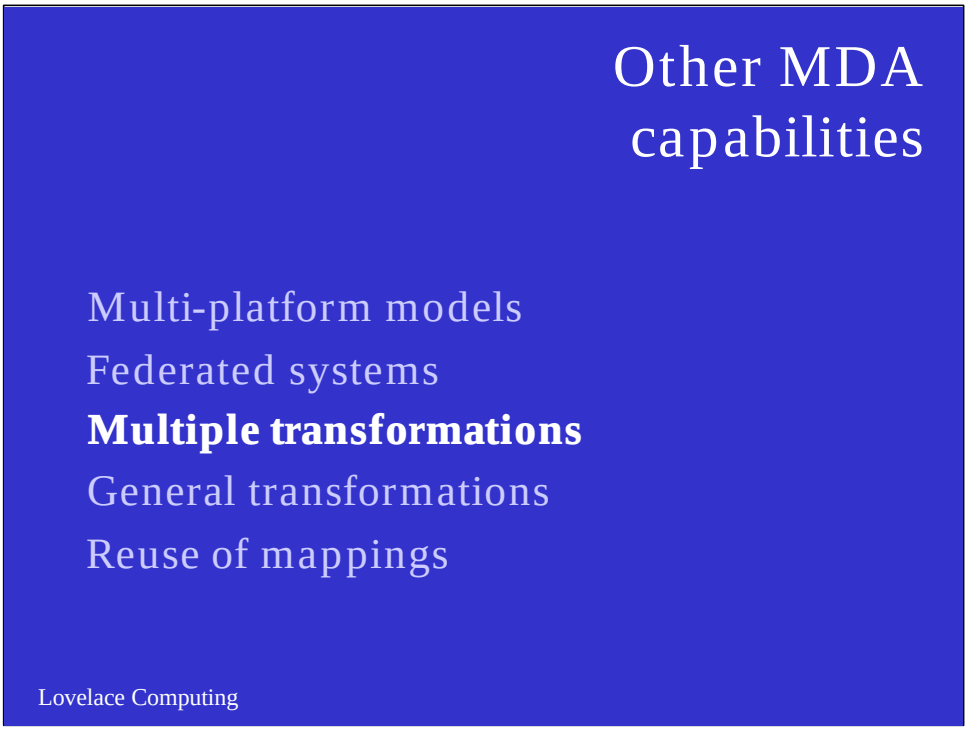
Reuse of mappings

Lovelace Computing

A PIM can specify a system, with several parts, each under separate control. The transformation of that PIM to a PSM can be made recognizing that the system is federated. That PIM can also be transformed into different PSMs for use by different parts of the system.

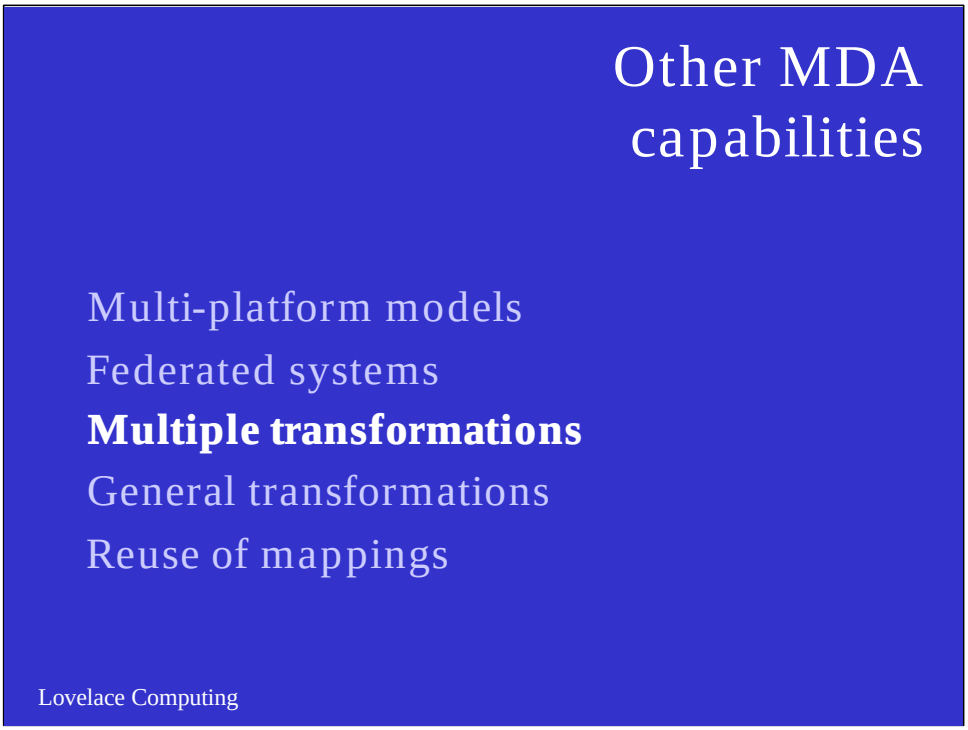
Example: Several trading partners want to share a common software design and produce interoperable implementations, each partner using a different platform.

This approach will require the identification of generic bridges between the platforms, or the generation of bridges specialized for the system. The use of platform independent models for specifying the whole system will provide generation tools with some, or most of the information needed to perform specific bridging, as long as a generic interoperability mechanism is available. No current standard solutions exist in this space. This is a topic for future standards in OMG.



The MDA pattern includes a PIM, a platform, and a PSM. The PSM is specific to that platform. The PIM is platform independent because it is not dependent on any particular platform of that class. What counts as a PIM depends on the class of platform that the MDA user has in mind.

Example: An OMG domain task force may be conducting an RFP process for a domain specific technology. It requests a PIM and a PSM for a generic component technology platform. At the same time, an OMG platform task force may be conducting an RFP process for an improved component model, backward compatible with the CORBA Component Model, CCM. This task force requests a PIM for a component technology and one or more PSMs for that technology. What is a PSM to the first task force is a PIM to the second.



The MDA pattern can be applied several times in succession. What is a PSM resulting from one application of the pattern, will be a PIM in the next application.

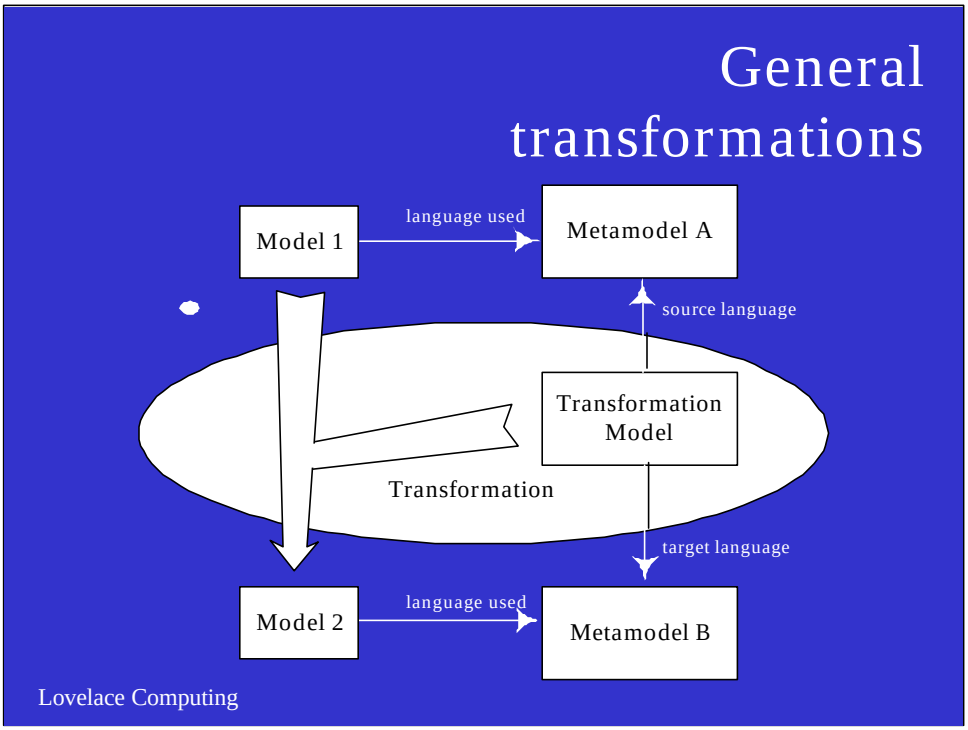
Example: In case of CORBA the platform is specified by a set of interfaces and usage patterns that constitute the CORBA Core Specification [CORBA]. The CORBA platform is independent of operating systems and programming languages. The OMG Trading Object Service specification [TOS] (consisting of interface specifications in OMG Interface Definition Language (OMG IDL)) can be considered to be a PIM from the viewpoint of CORBA, because it is independent of operating systems and programming languages. When the IDL to C++ Language Mapping specification is applied to the Trading Service PIM, the C++-specific result can be considered to be a PSM for the Trading Service, where the platform is the C++ language. Thus the IDL to C++ Language Mapping specification [IDLC++] determines the mapping from the Trading Service PIM to the Trading Service PSM.

Other MDA capabilities

- Multi-platform models
- Federated systems
- Multiple transformations
- **General transformations**
- Reuse of mappings

Lovelace Computing

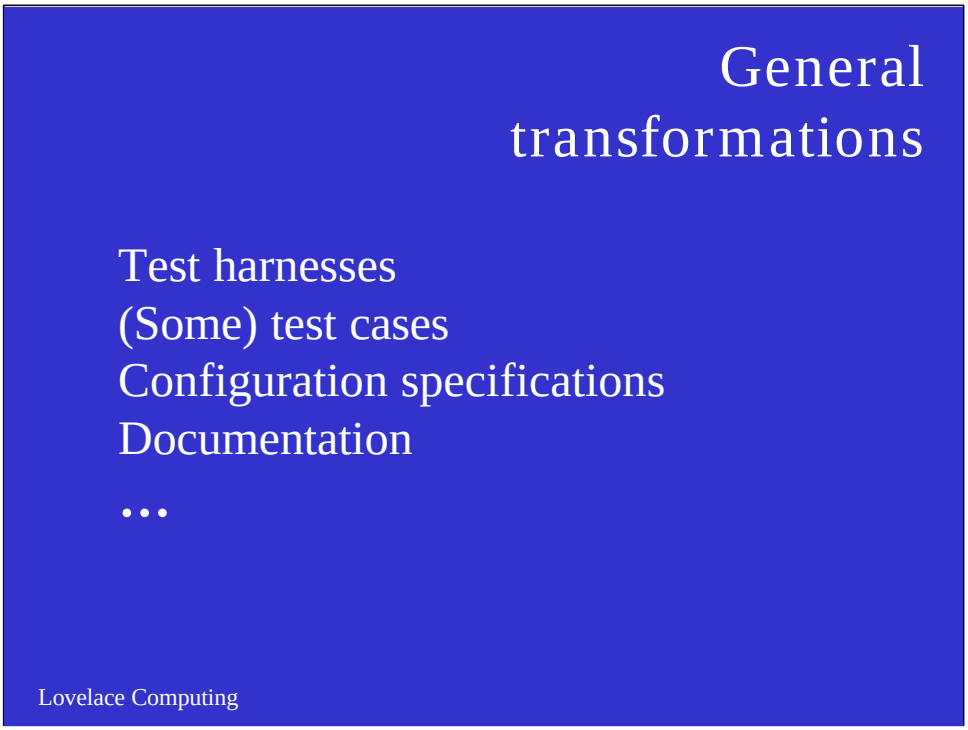
The same approaches that enable transformation of a PIM to a PSM can be used to transform any model into another, related model.



The drawing illustrates the general case of a metamodel mapping transformation. Model 1 and Model 2 may be any models, and the transformation need not have anything to do with platforms.

Examples: A generic model of financial transactions is transformed to one specific to a particular kind of transaction. A generic model of financial transactions is transformed to one specific to the trade practices of a particular exchange. An internationalized model of an application is transformed to one specific to the customs of a particular region.

The drawing and example use metamodel mapping to illustrate the point. Any of the MDA approaches discussed in this Guide can be used for general model-to-model transformations.



The drawing illustrates the general case of a metamodel mapping transformation. Model 1 and Model 2 may be any models, and the transformation need not have anything to do with platforms.

Examples: A generic model of financial transactions is transformed to one specific to a particular kind of transaction. A generic model of financial transactions is transformed to one specific to the trade practices of a particular exchange. An internationalized model of an application is transformed to one specific to the customs of a particular region.

The drawing and example use metamodel mapping to illustrate the point. Any of the MDA approaches discussed in this Guide can be used for general model-to-model transformations.

Other MDA capabilities

Multi-platform models

Federated systems

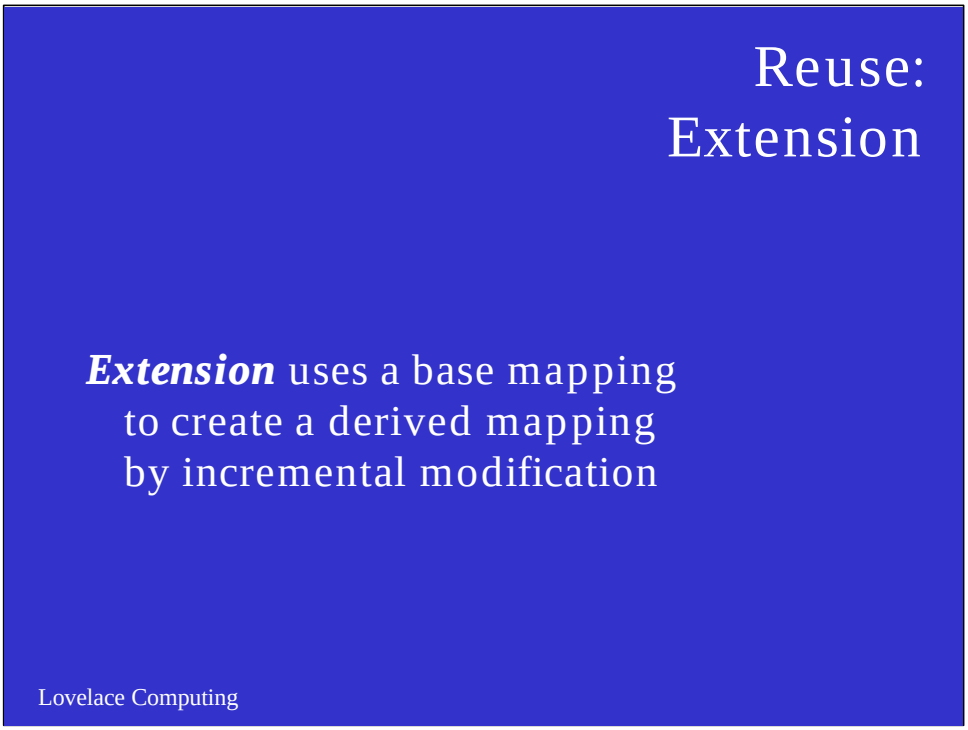
Multiple transformations

General transformations

Reuse of mappings

Lovelace Computing

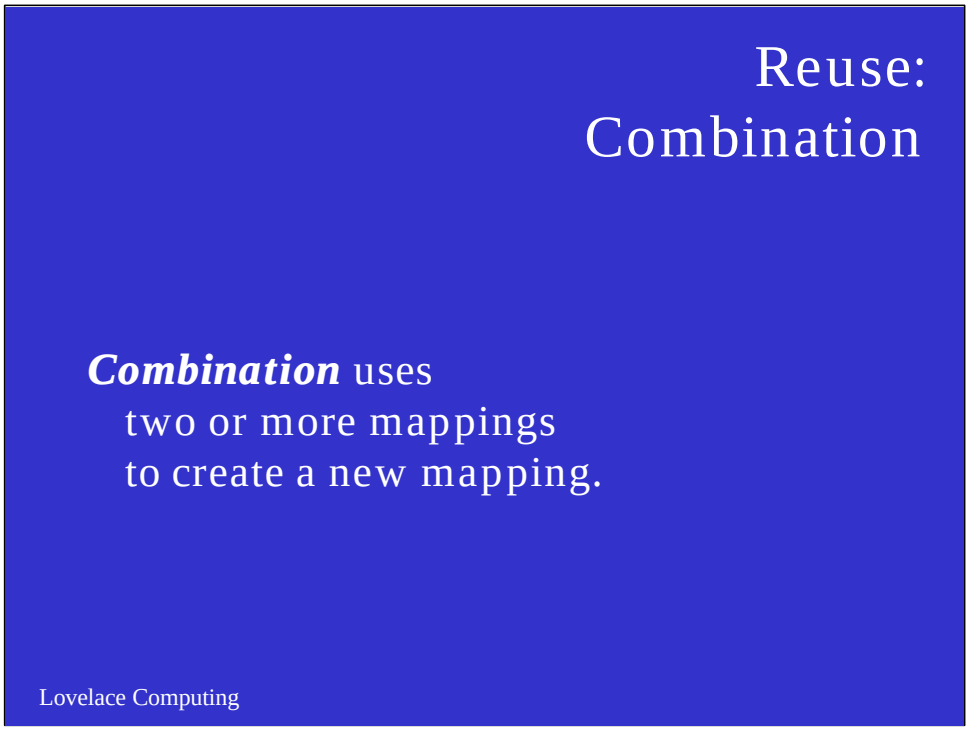
Mappings may be reused in several ways. These include extension, combination, and bridging.



Extension uses a base mapping to create a derived mapping by incremental modification. The incremental modifications may add to or alter the properties of the base mapping to obtain the derived mapping.

Mappings can be arranged in an inheritance hierarchy according to derived base mapping relationships. This is the interpretation of mapping inheritance in the MDA. If mappings can have several base mappings, inheritance is said to be multiple. If the criteria prohibit suppression of properties from the base mappings, inheritance is said to be strict.

Example: Given a mapping from UML class diagrams to generic CORBA models, the mapping can be extended to make a mapping for a specific vendor of a CORBA system.



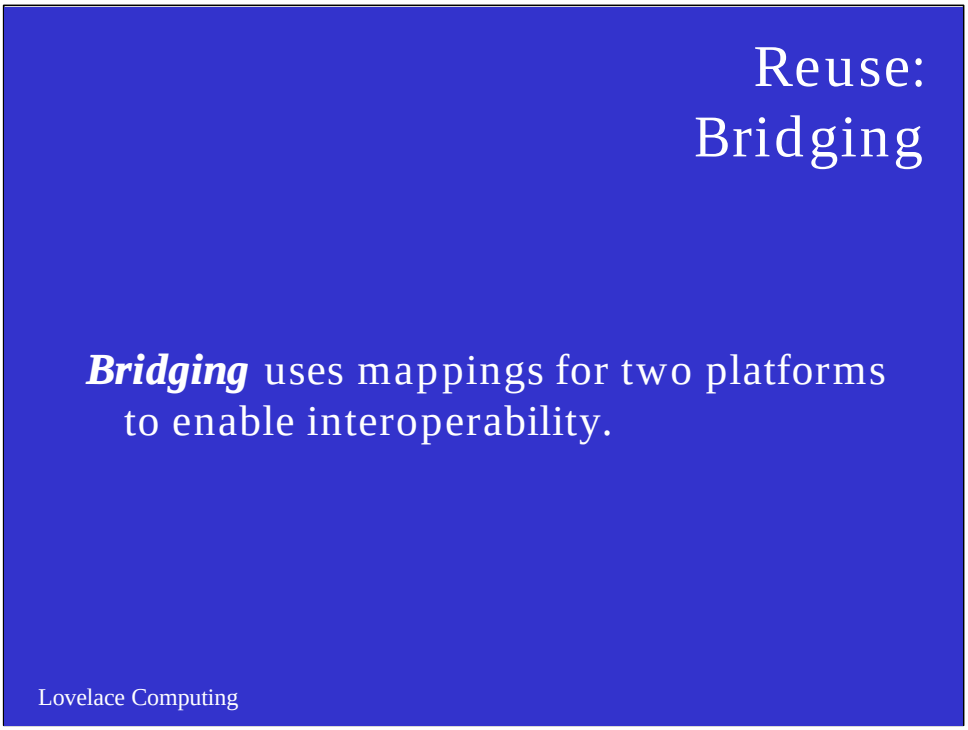
Combination uses two or more mappings to create a new mapping. The characteristics of the new mapping are determined by the mappings being combined and by the way they are combined. The effect of the application of a combined mapping is the corresponding combination of the effects of the original mappings.

Ways in which mappings may be combined include sequential combination and concurrent combination. The concept of a combination of mappings will always be used in a particular sense, identifying a particular means of combination.

Examples:

Given a mapping from platform independent models to component style models and a mapping from component style models to EJB code. A sequential combination applies the mappings successively to produce a mapping from PIM to EJB code. If instead, the second mapping is for transforming component style models to CORBA Component Model code, the sequential combination is for transforming PIMs to CCM code.

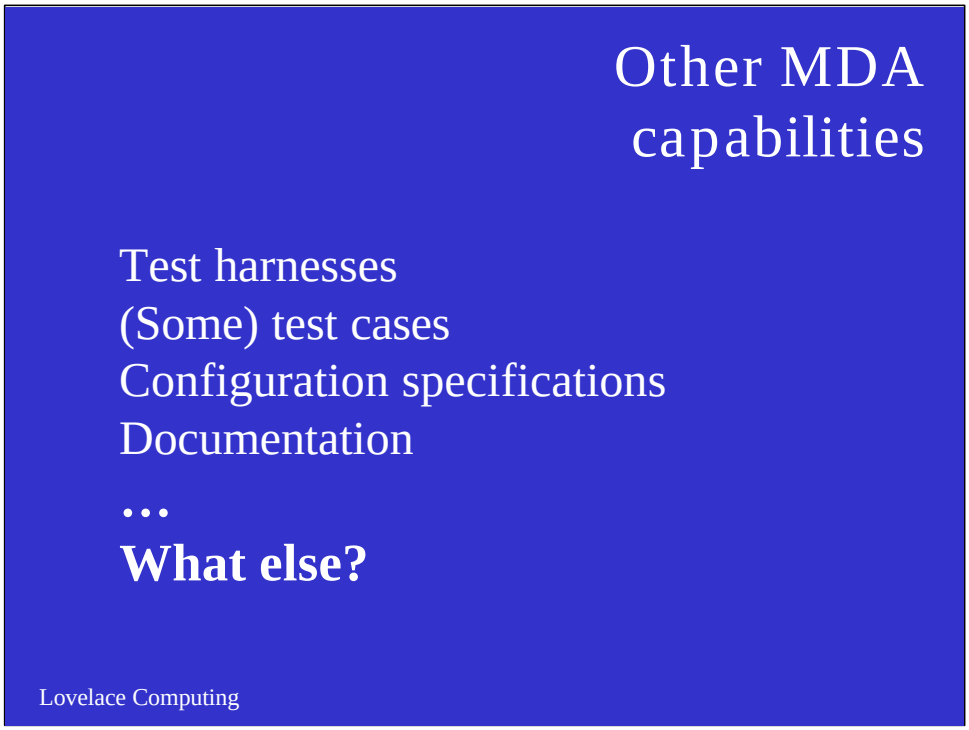
Given a mapping from PIMs to CCM specific models, which includes a mark for container managed persistence and a mark for component managed persistence and another mapping from PIMs to high performance and high availability indexed sequential file access. A concurrent combination applies both of the mappings concurrently to produce a PSM in which some objects use CCM persistence services and others use the file access platform.



An interoperability mapping uses mappings for two different platforms. These are combined to create a mapping to transform a PIM into a PSM in which some objects are on one platform and others on the second. This mapping is then extended further to include connectors that bridge between the two platforms and specifications for the use of these connectors in a transformation. The resulting mapping is used to transform a PIM into a PSM of a system that makes use of both platforms and provides for the interoperability of the subsystems on the different platforms.

Other MDA capabilities

- Multi-platform models
- Federated systems
- Multiple transformations
- General transformations
- Reuse of mappings

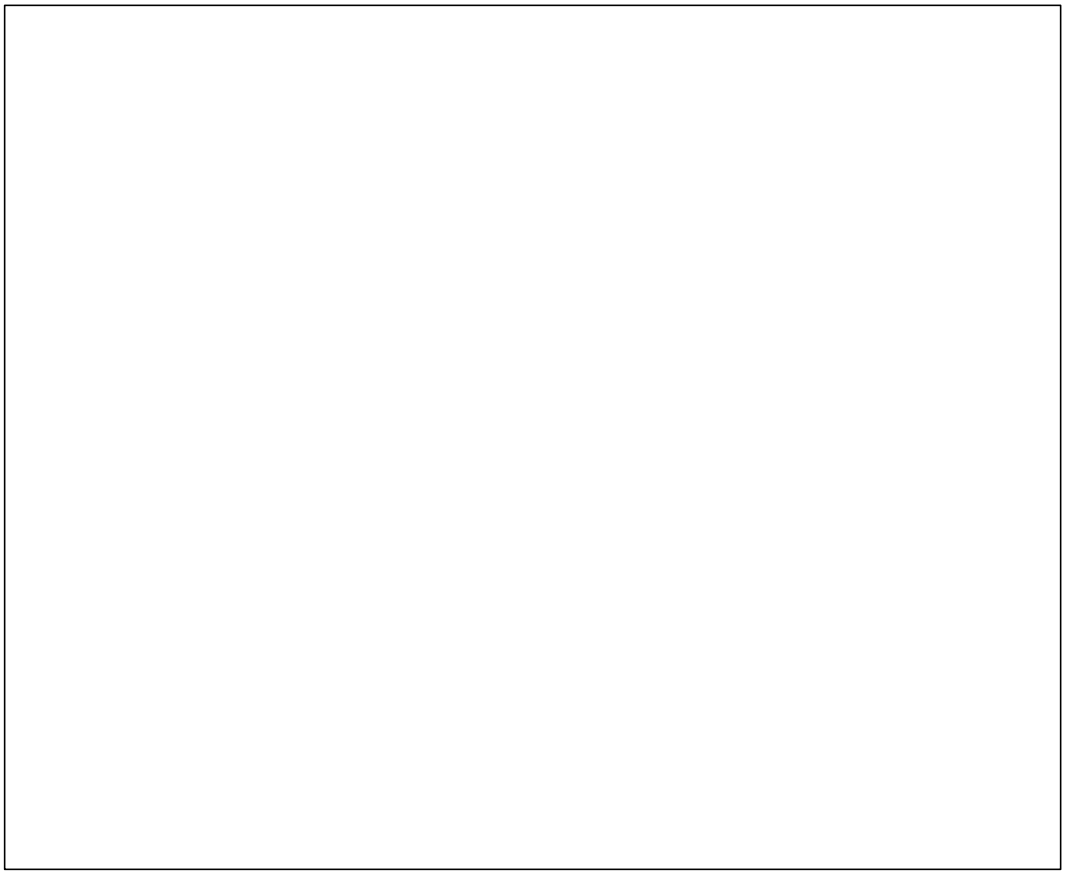


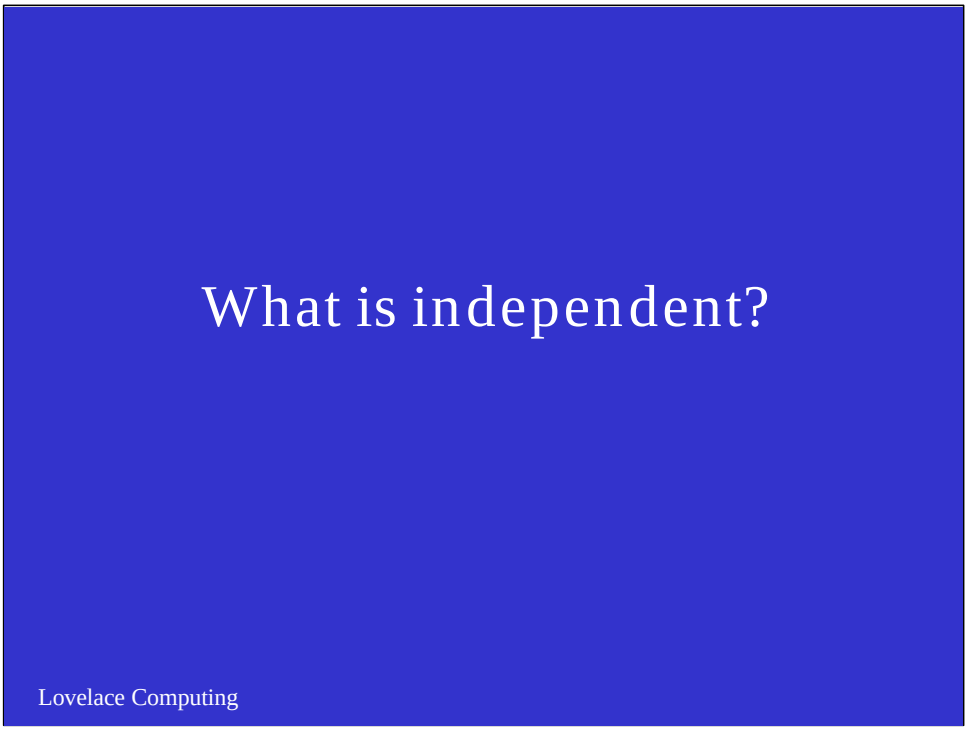
Transformations can produce many kinds of results.

UML, an Action Language, and OCL are needed as a package, if there is to be code generation, instead of just skeleton generation.

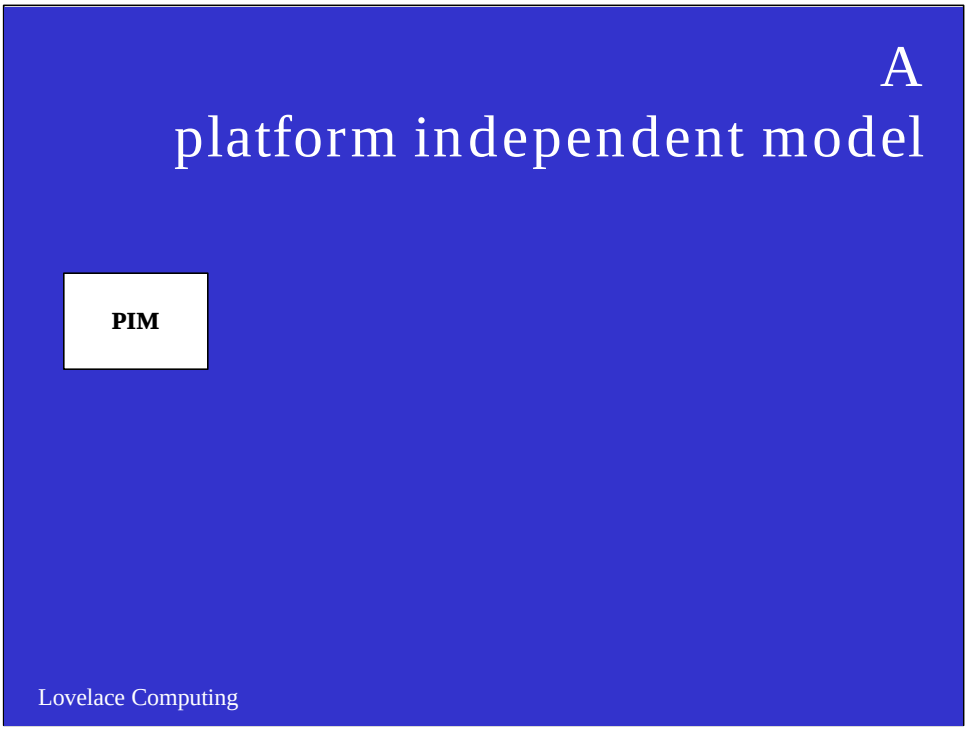
Using the MDA pattern

Lovelace Computing

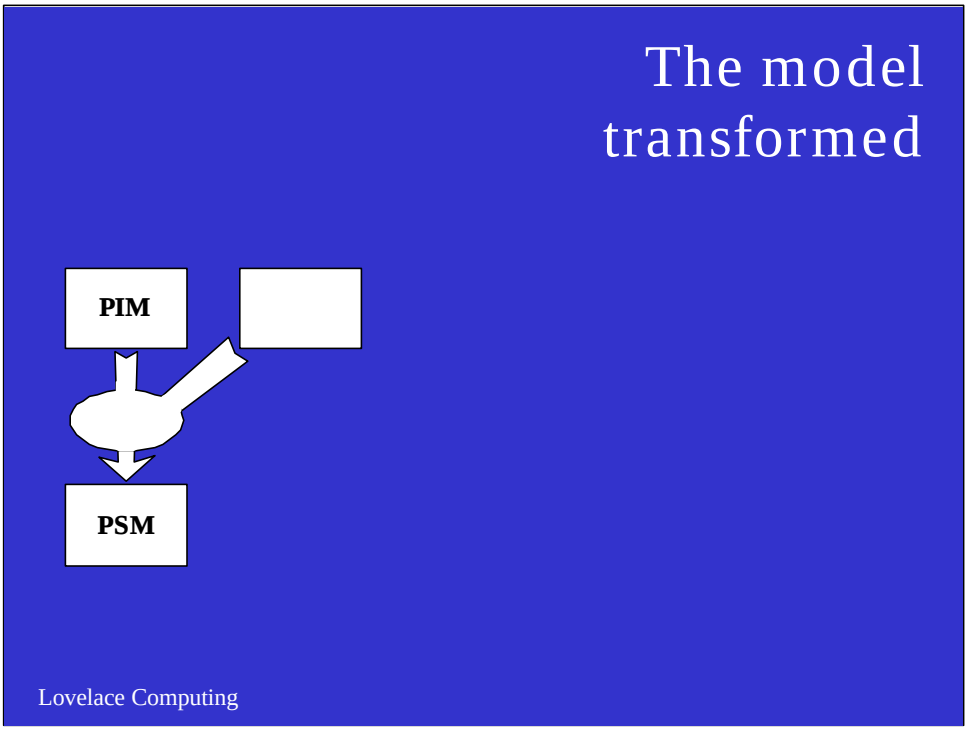




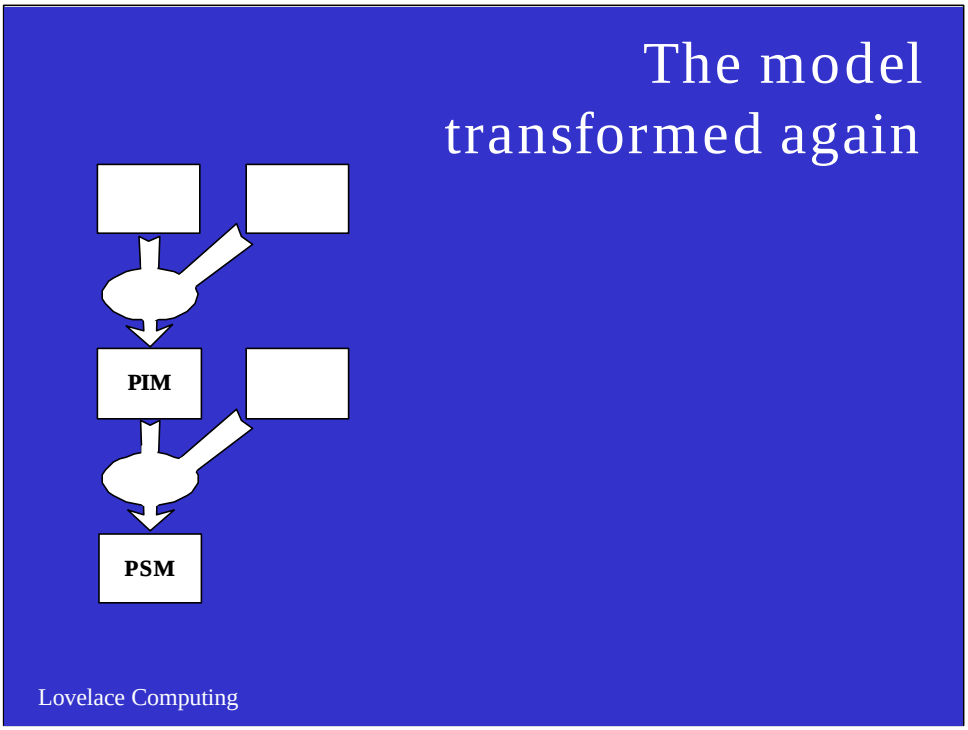
The MDA pattern may be applied more than once.



The original PIM is an application model, designed to be independent of many platform choices.

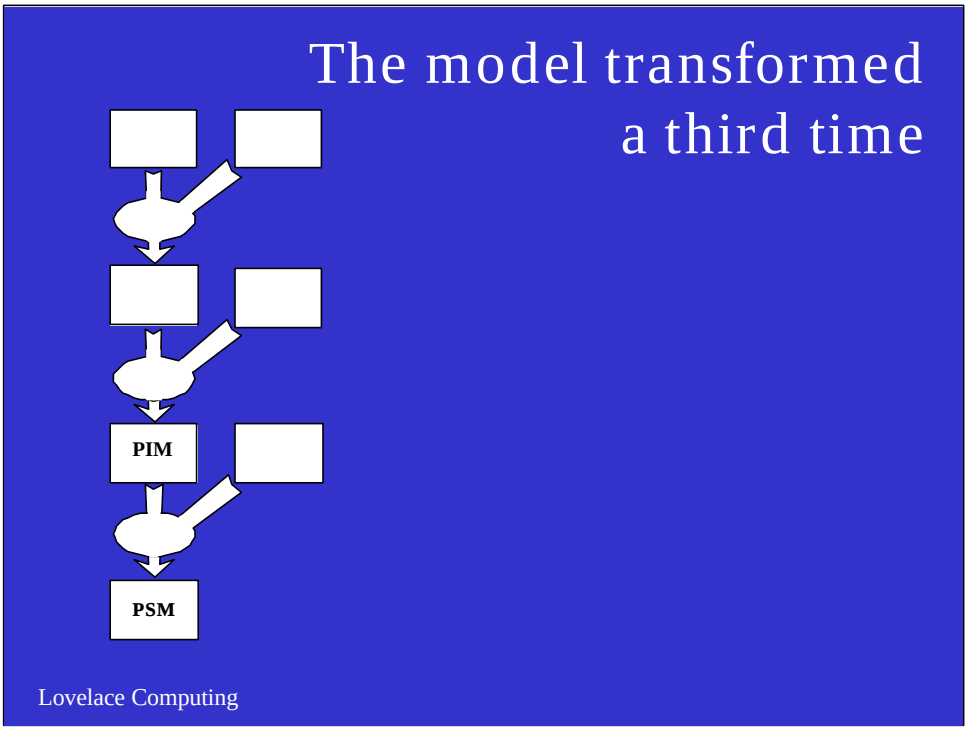


It is transformed to a PSM specific to component platforms. But the transformation has been carried out so that the model remains independent of the choice of a particular component platform.

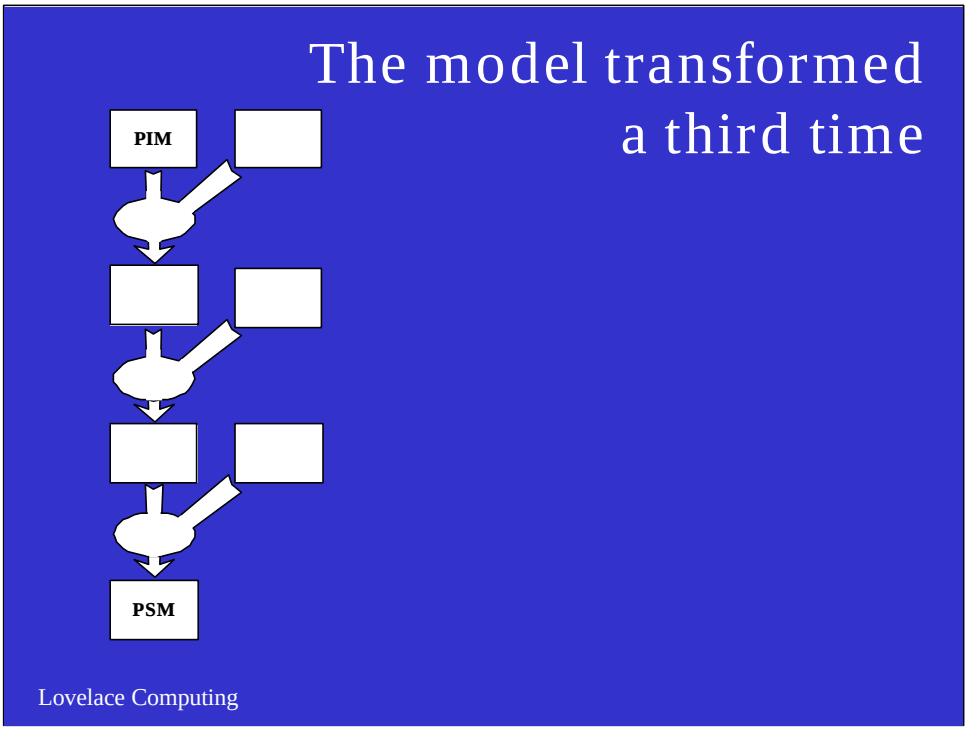


The MDA pattern is applied again.

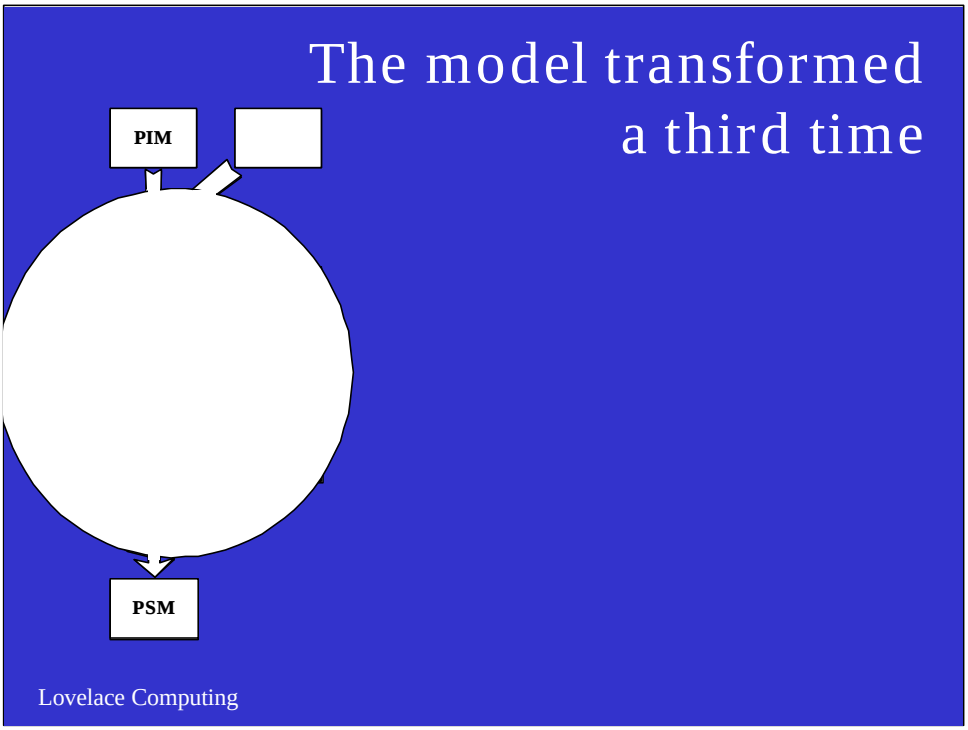
The model in the role of PSM in the first transformation is in the role of PIM in the second transformation. The resulting PSM is specific to CORBA Components.



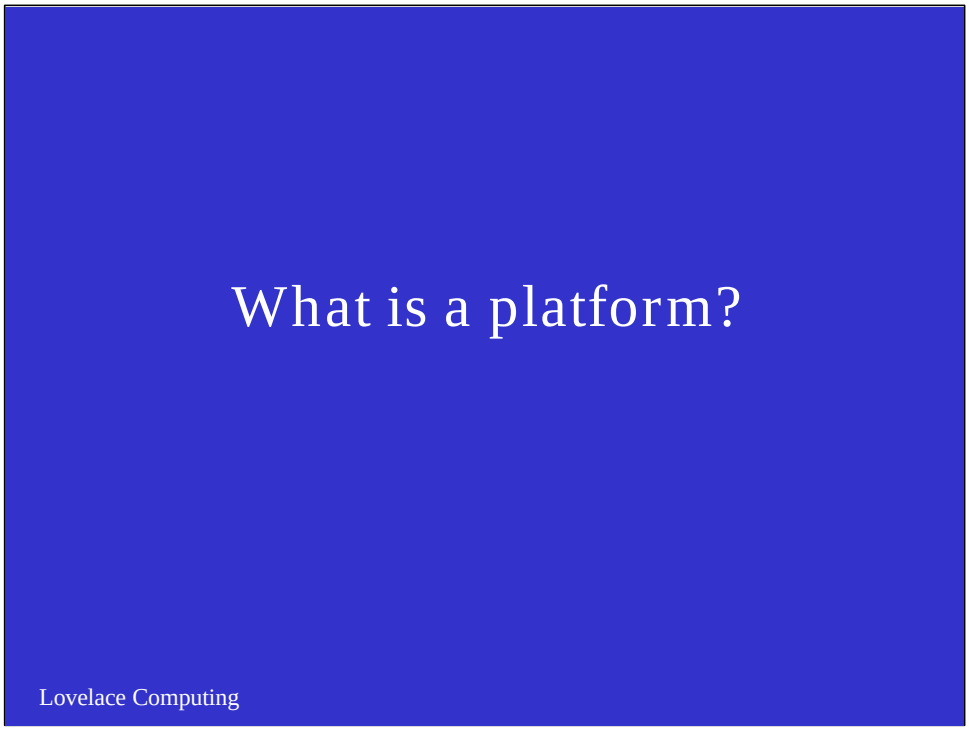
It may be desirable to transform this model again, to make specialized use of the platform in order to achieve a certain quality of service, perhaps to meet an availability requirement.



The original PIM, after three transformations, gives a PSM for high availability on a CORBA Components platform.



This can be seen as a single transformation

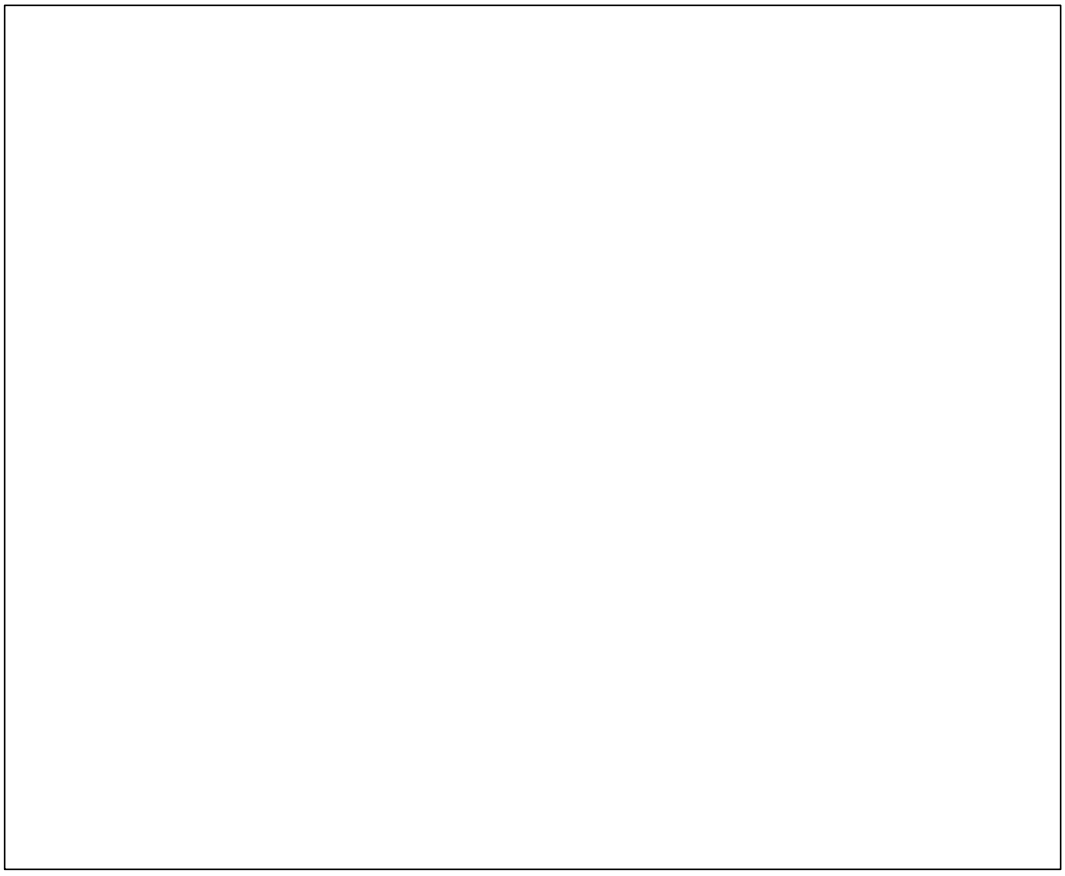


Serial transformations of this sort may or may not be common in practice. The example does, however, raise an altogether different question:

Wait a minute, just what counts as a platform, exactly?

Some notation

Lovelace Computing



Some notation

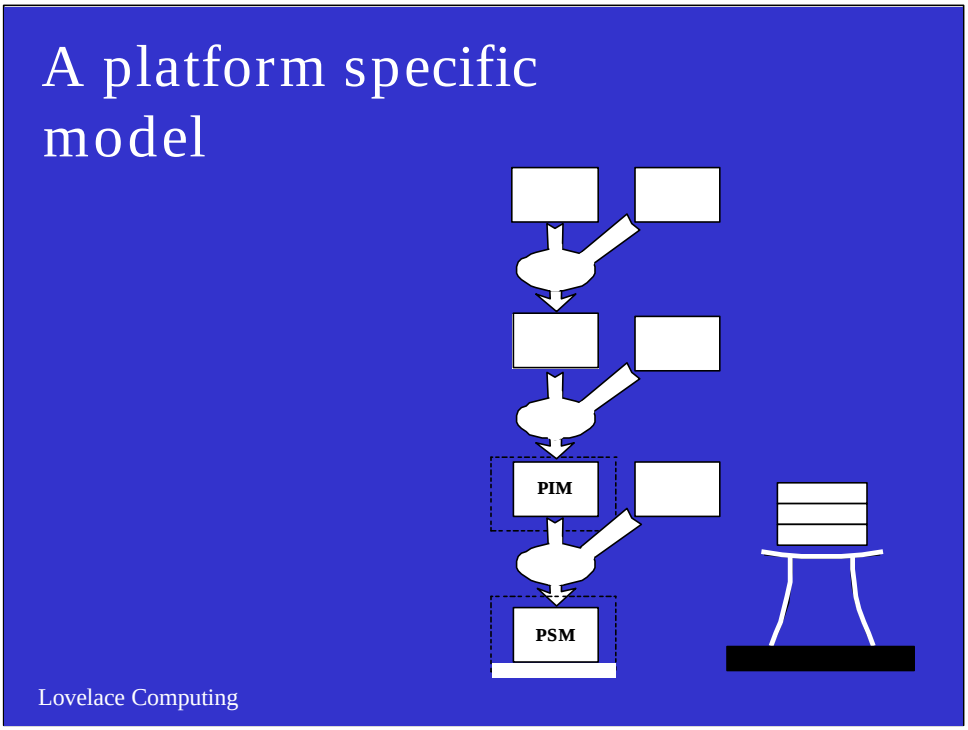
An application

A system consists of one or more ***applications***, supported by one or more platforms

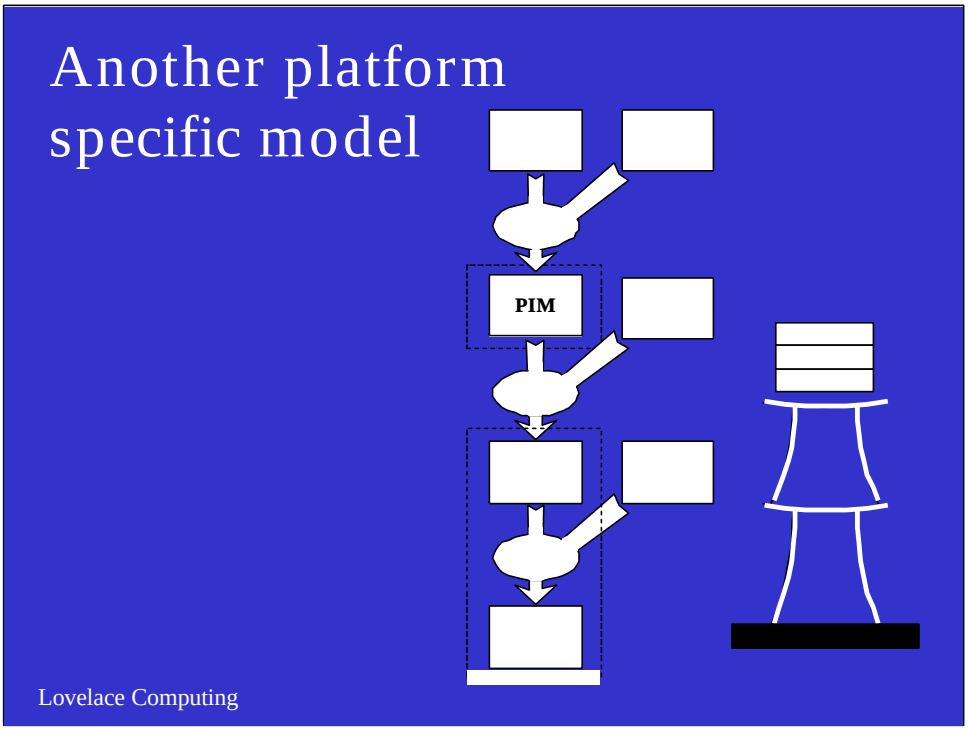
Lovelace Computing

Some notation

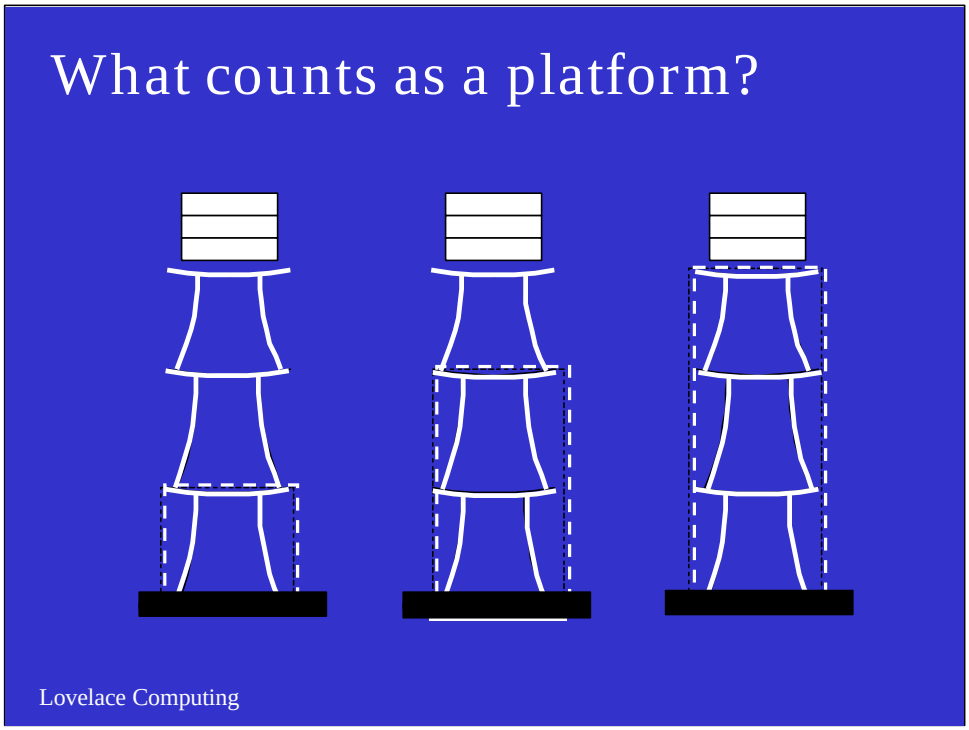
A platform



The PIM on the left is a model of the application on the right; this model is in the platform independent role. The PSM on the left is a platform specific model of the application, for the platform shown on the right.



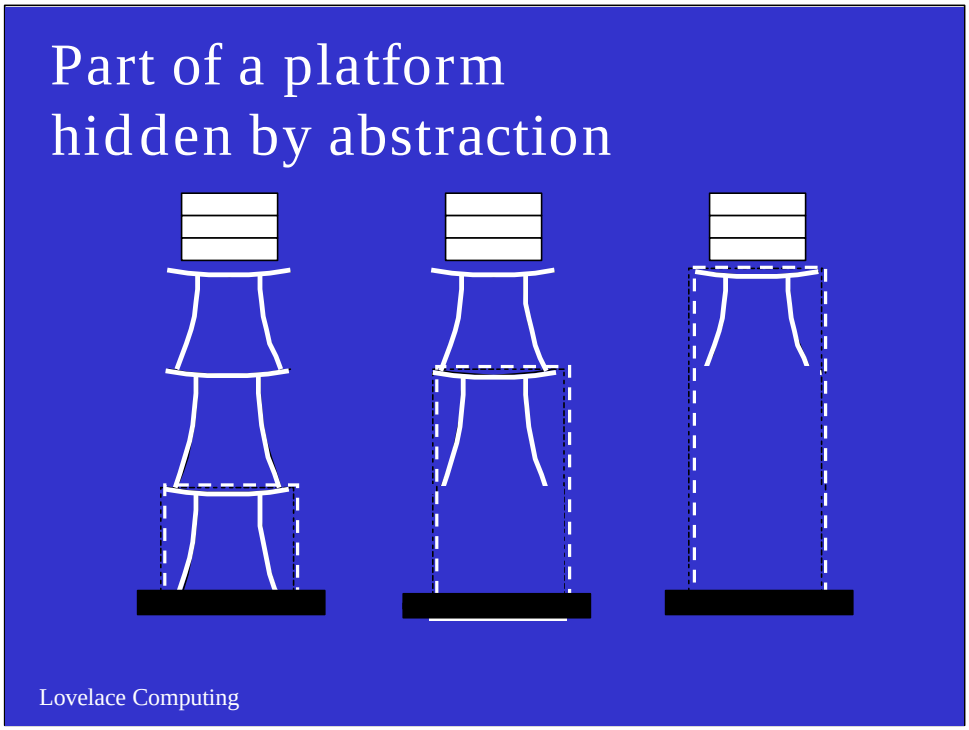
This is the same application and platform, from a different viewpoint.



To be adopted, a submitted technology must include a PIM and at least one PSM; in addition, there must be an implementation or a commitment to provide an implementation within a year. These are three different views of the same application with its platform. The dashed lines enclose the parts of the technology that are, from the different viewpoints, considered to be the implementation of a platform.

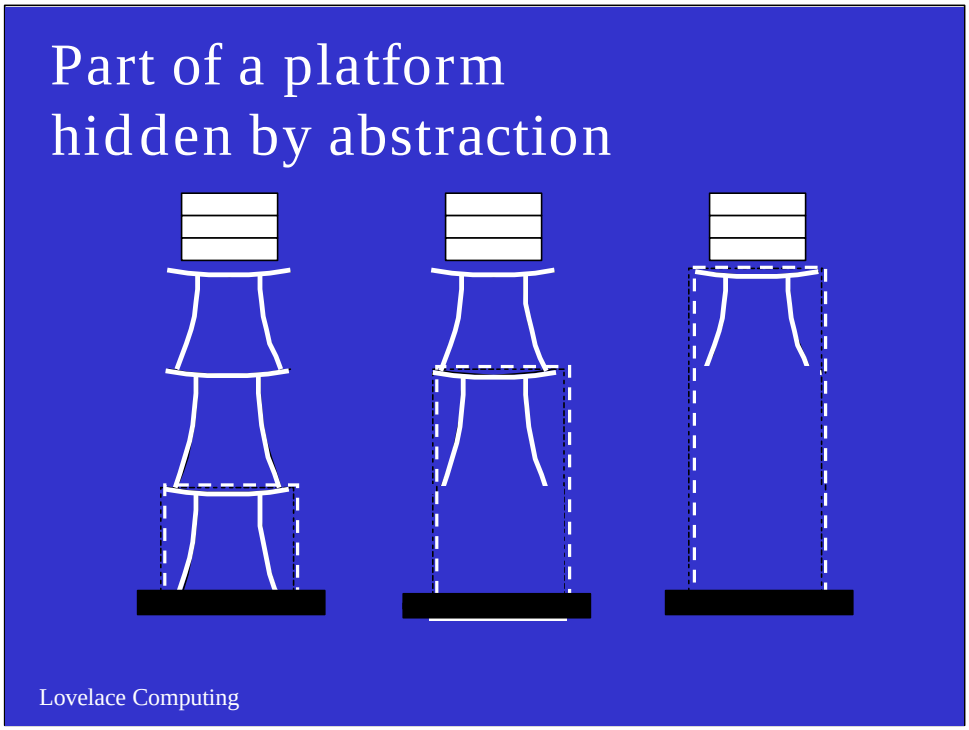
Which viewpoint is taken depends on the needs of the user of the model.

Any of the parts of the model enclosed in the dashed line may be considered to be the platform. Wherever it is considered to start, the platform goes all the way down to a complete implementation.



A PSM is not required to include all details of the platform. But, by definition, an implementation must "provide the information needed to create an object and to allow the object to participate in providing an appropriate set of services."

In the illustration, some of the details of the platform that supports the application are hidden. For example, a PSM specific to the CORBA platform may hide the details of the programming language and operating system. A PSM specific to CORBA Components may hide the details of CORBA along with the programming language and operating system.

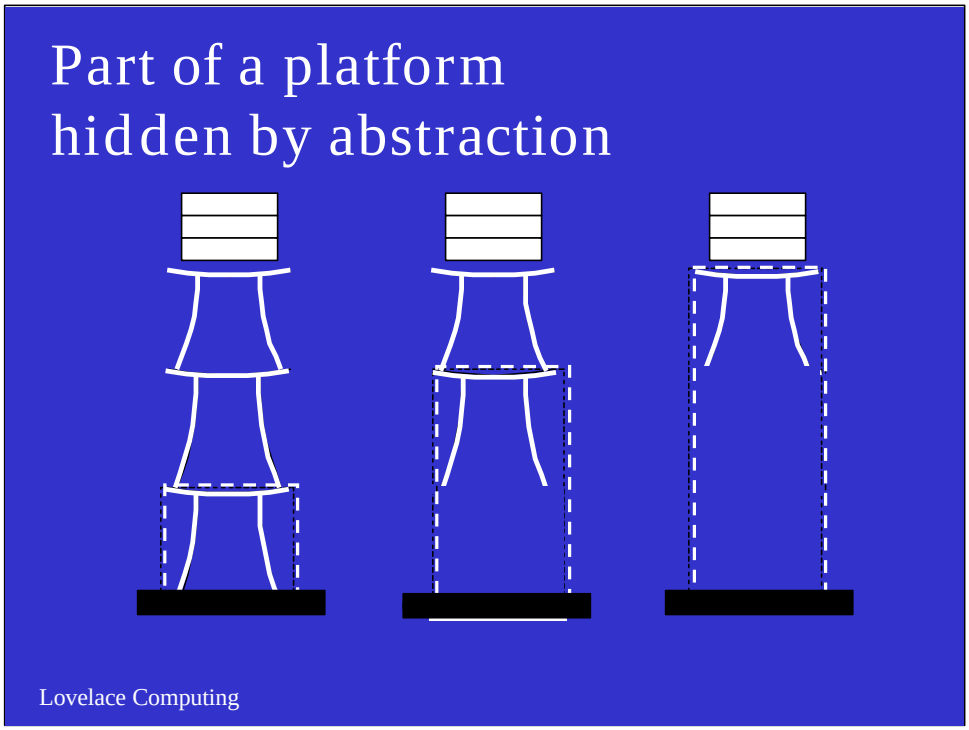


When a platform provides a degree of portability, it is appropriate to hide the details of the particular supporting platform, since portability makes it possible to choose one or another supporting platform.

The entire platform or set of platforms is there in an implementation, even if hidden in a PSM.

To repeat this: a PSM may or may not include a detailed model of the platform. If it does not, either it is an abstract model, that hides those details, or it makes reference (explicit or implicit) to another model or models that provide the details. It is not a PSM unless it can be used to produce an implementation. So it must include all details necessary for an implementation, or those details must be included by reference.

Suppose, for example, that a PSM is specific to CORBA. Then it need not include all the details necessary to implement CORBA, because it makes implicit (or better yet, explicit) reference to the specifications of those CORBA capabilities it uses. Either these specifications are available to complete the PSM, or actual platforms are available which will provide the support required to complete the implementation (in the case of CORBA, both).

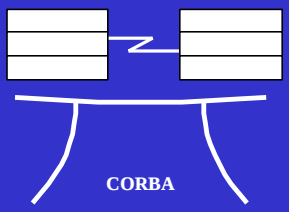


What counts as a platform depends on the kind of system being developed.

Example: From the point of view of a developer of middleware for several operating systems, there will be platform independent and platform specific models of the middleware. The class of platforms is the operating systems and each target platform is a particular operating system.

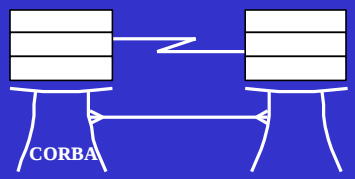
What counts as a platform is relative to the purpose of the modeler. For many MDA users, middleware is a platform, for a middleware developer an operating system is the platform. Thus a platform-independent model of middleware might appear to be a highly platform-specific model from the point of view of an application developer

Interoperability: a common platform



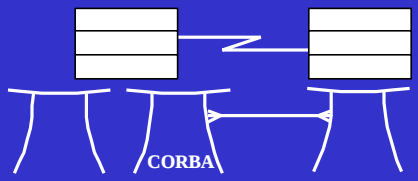
Lovelace Computing

Interoperability: a bridge



Lovelace Computing

Interoperability: another case



Lovelace Computing

This presentation and your workshop CD

For a version of this presentation more complete and up to date than the one on your CD, have a look at:

www.joaquin.net/MDA/

Lovelace Computing

This presentation and your workshop CD

The version used in the workshop
is there at the time of the workshop

By two weeks after the workshop
I hope to have another version,
based on your feedback here today.

www.joaquin.net/MDA/

Lovelace Computing

MDA

Model Driven Architecture

Joaquin Miller
Lovelace Computing
representing X-Change Technologies

This is a presentation prepared for the OMG Fourth Workshop On UML™ for Enterprise Applications: Delivering the Promise of MDA

Joaquin Miller

Lovelace Computing Company
www.joaquin.net