What's a Platform?

An Overview of Model Driven Architecture

Joaquin Miller
Lovelace® Computing
representing X-Change Technologies

This presentation discusses the concepts of Model Driven Architecture.

An important contribution of the presentation is to elucidate the intended meanings of the central terms, 'platform' and 'independence,' as they are used in the OMG documents.

It will benefit the workshop to establish for the workshop participants a common understanding of MDA concepts and a shared usage of terminology. This presentation is designed to help accomplish that.

These slides also appear in the workshop tutorial: Model Driven Architecture.

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

Independent of What?

An Overview of Model Driven Architecture

Joaquin Miller
Lovelace® Computing
representing X-Change Technologies

I like this title. I proposed it to the program committee. It suits my style. But then, the intended meaning might not be clear to all.

What's a Platform?

An Overview of
Model Driven Architecture

Joaquin Miller
Lovelace® Computing
representing X-Change Technologies

Acknowledgements

The material in this presentation is largely from the OMG MDA Guide.

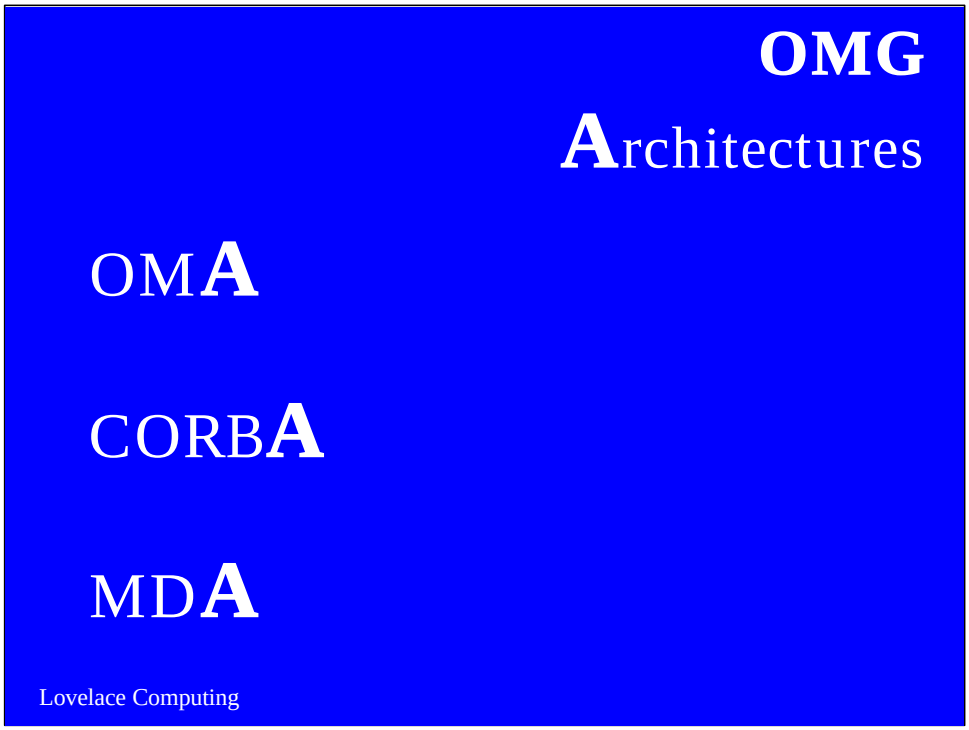
The Guide was prepared by the ORMSC, under the supervision of the AB.

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)

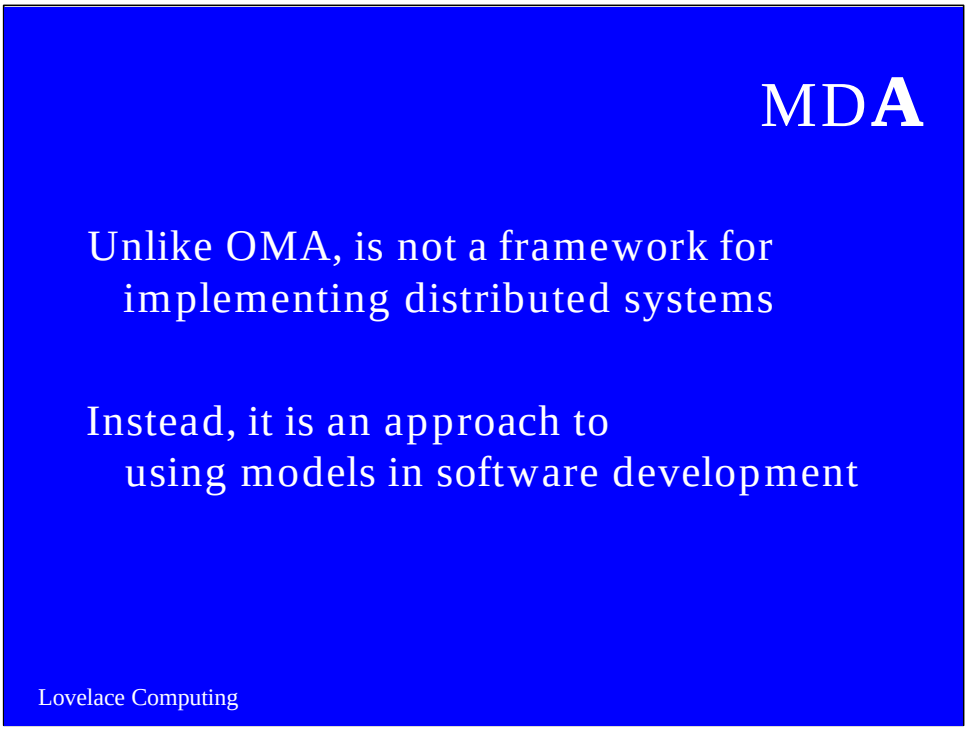


Over the last dozen years, the Object Management Group, better known as OMG, standardized the object request broker (ORB) and a suite of object services. This work was guided by the Object Management Architecture (OMA), which provides a framework for distributed systems and by the Common ORB Architecture,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



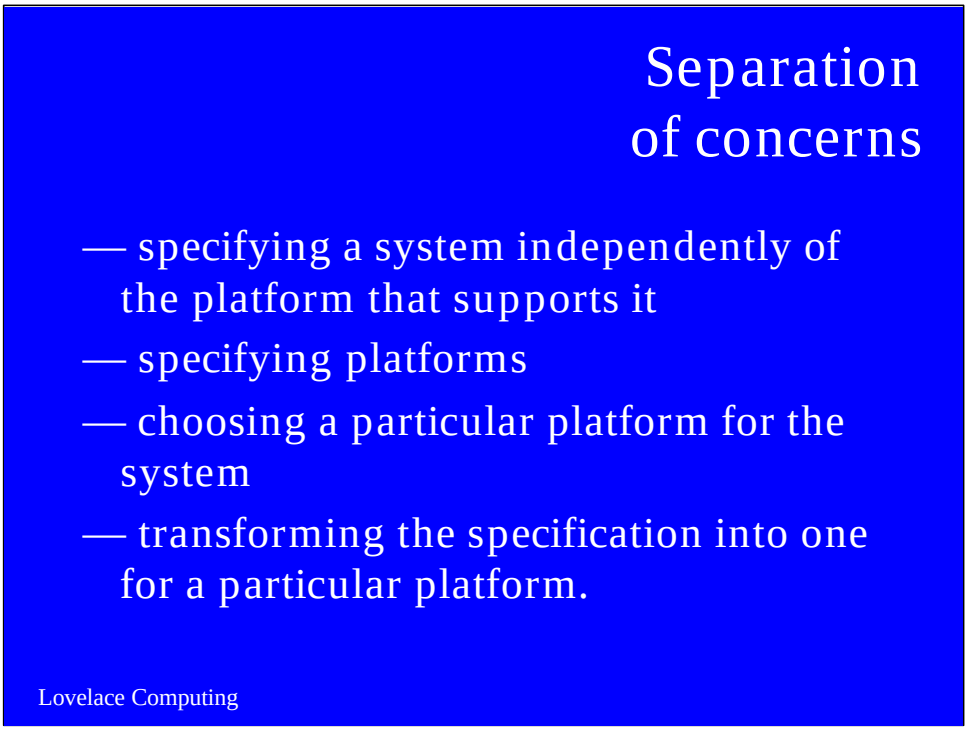
MDA is not, like the OMA and CORBA, a framework for implementing distributed systems. It is an approach to using models in software development.

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

Three primary goals of MDA are

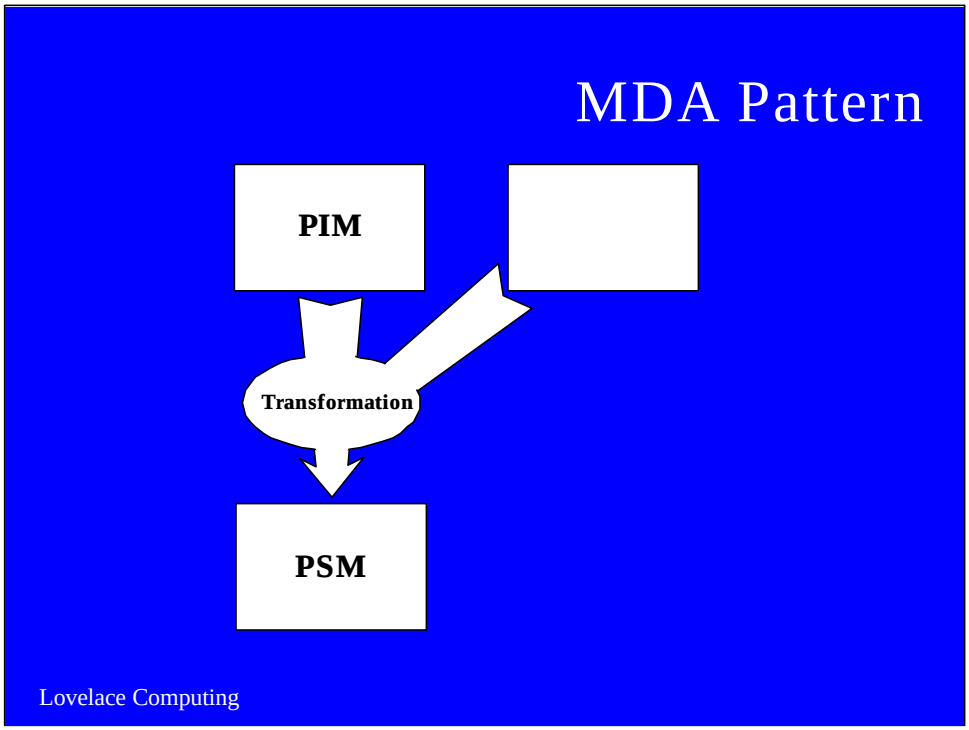
— portability,
— interoperability and
— reusability

Two other goals are



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.

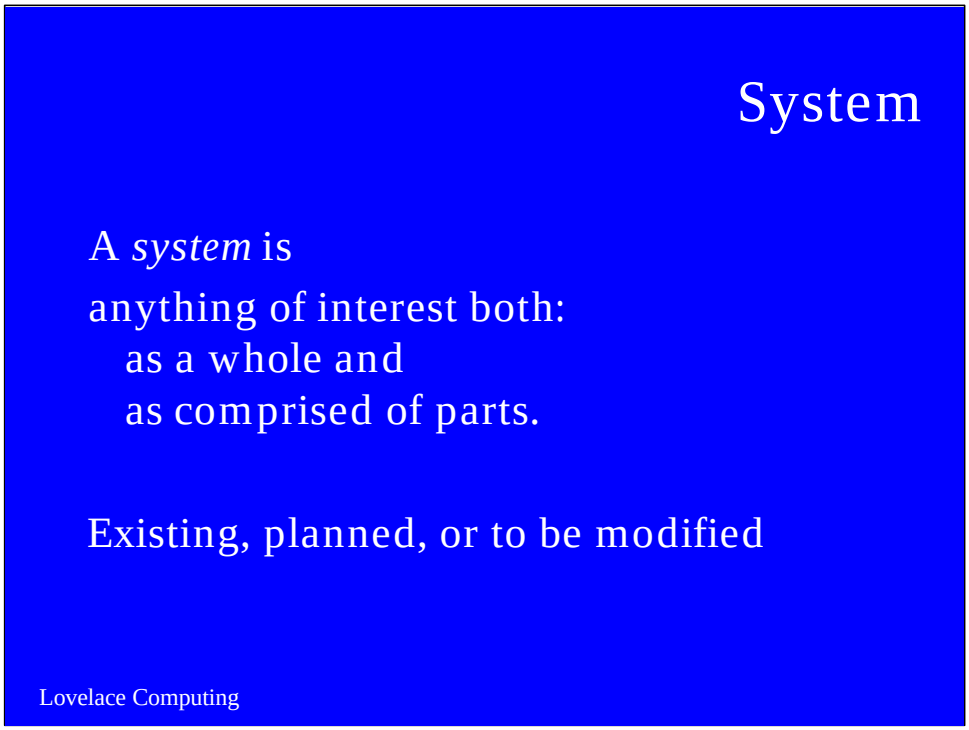
The drawing is intended to be suggestive. The platform independent model and other information are combined by the transformation to produce a platform specific model.

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

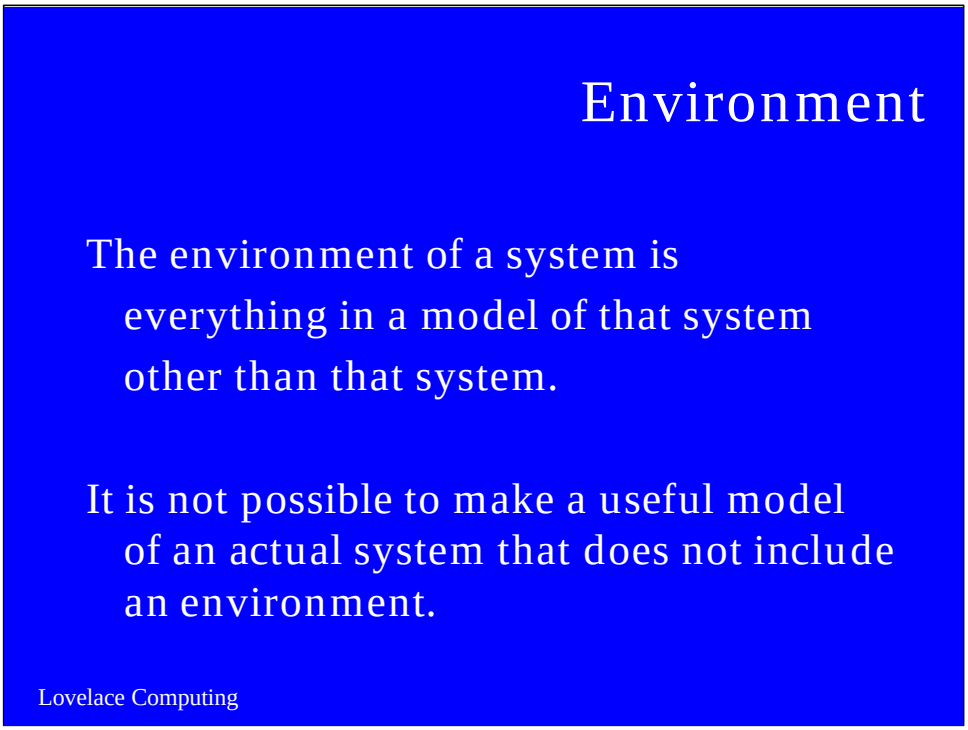


We'll present the MDA ideas in terms of some existing or planned system. That *system* may include anything: a program, a single computer system, some combination of parts of different computer systems, a federation of computer systems, each under separate control, people, an enterprise, a federation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

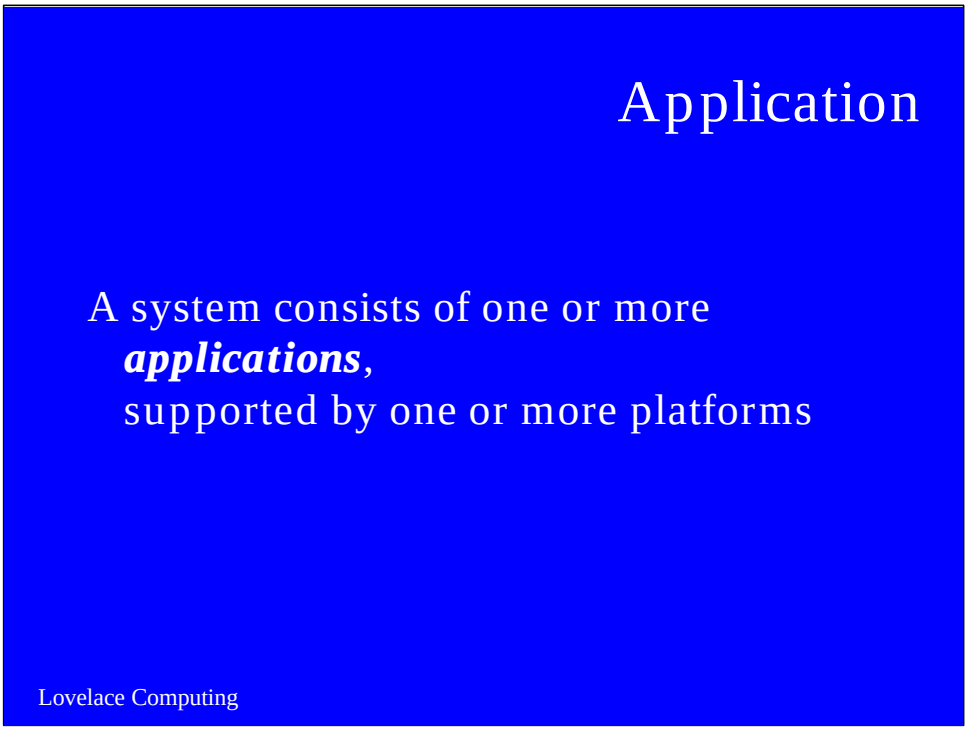
Of course, the system and its environment form another system. When the distinction does not matter, it is not this larger system we mean in this presentation, but the system being specified or described.

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.

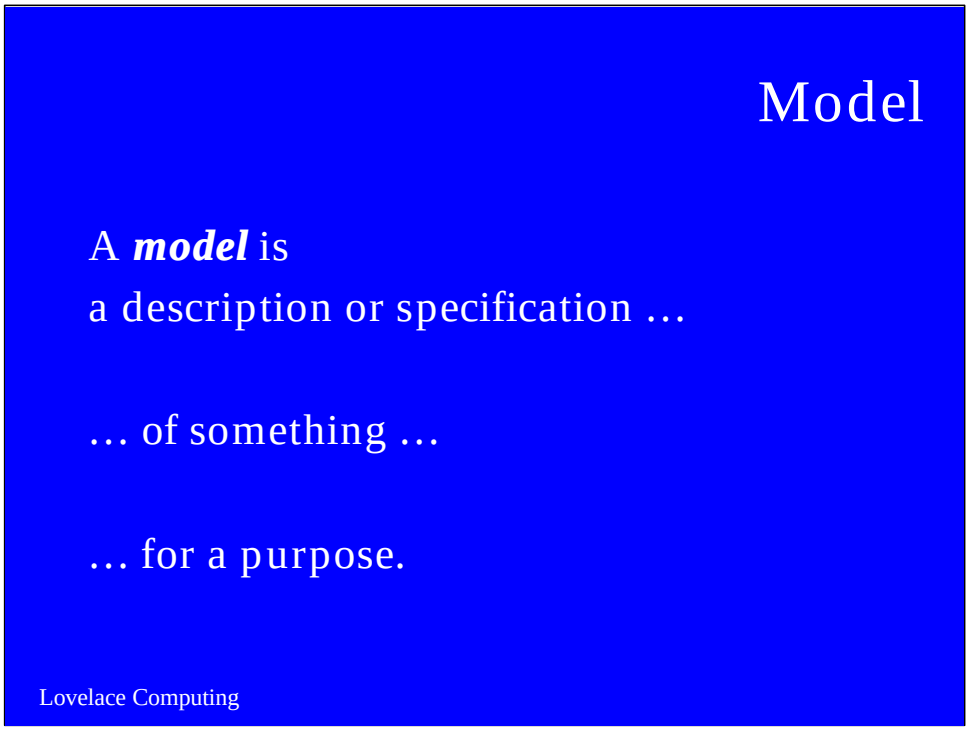
The reason that the statement is not exactly true is that the universe is a closed system, so has no environment.

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



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

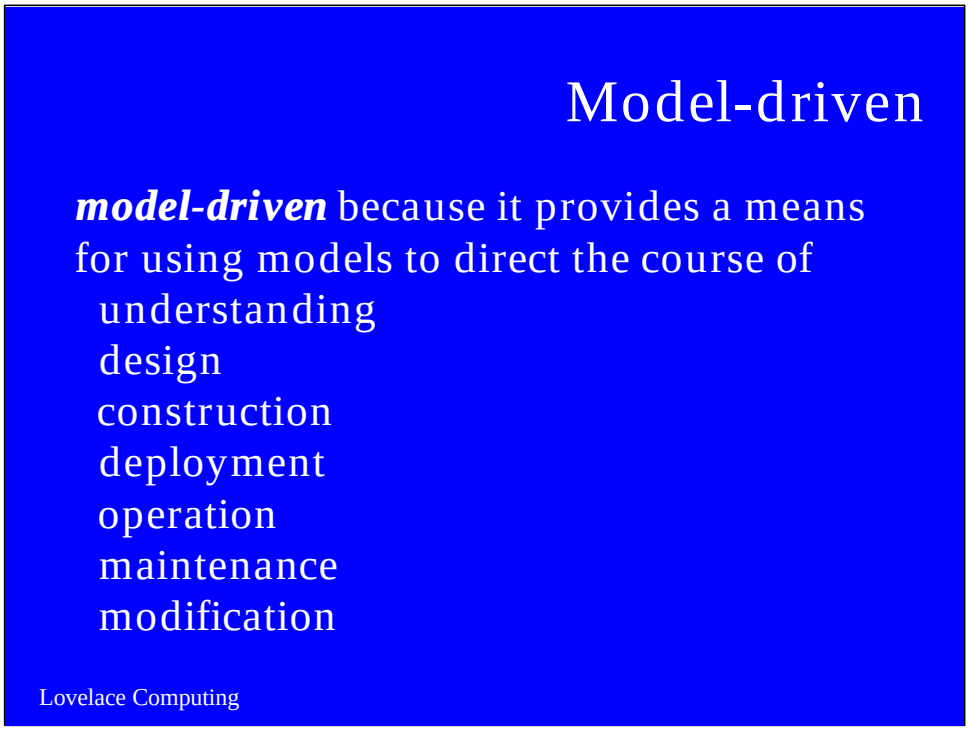
Model

A model is
a description or specification …

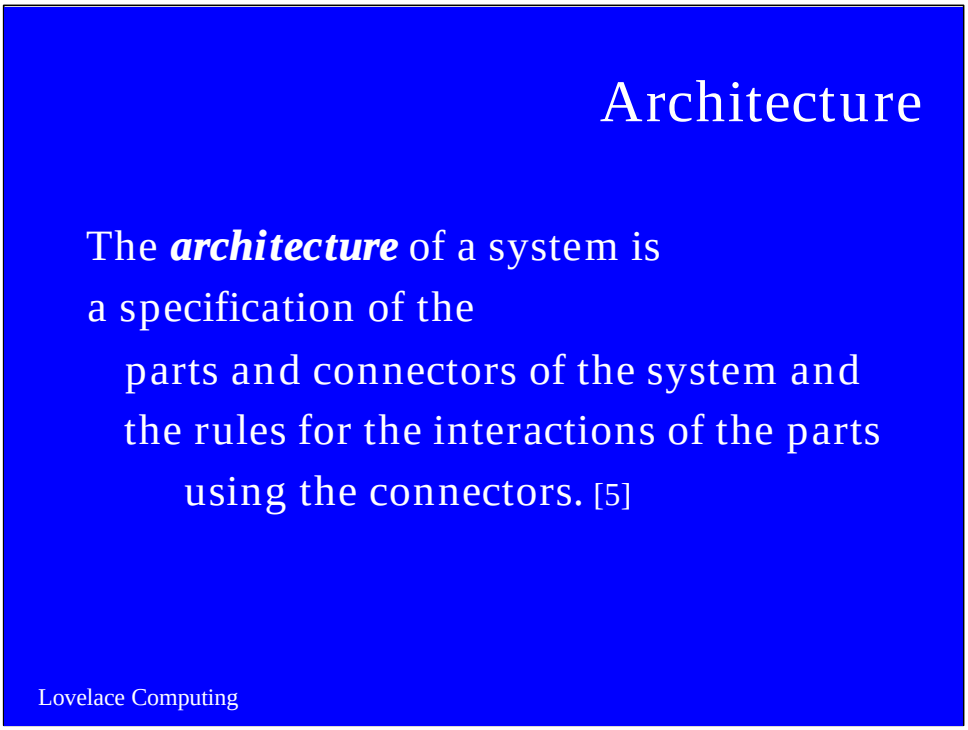
… of something …

… 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.

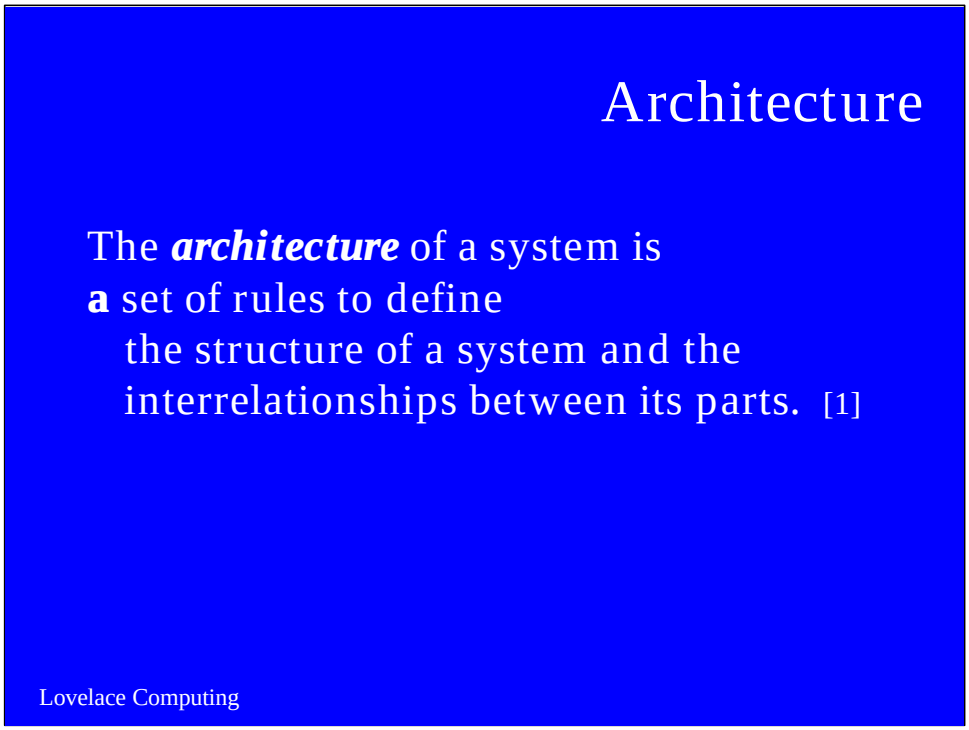
[RM-ODP 2-6.6 www.joaquin.net/ODP/Part2/6.html#6.6]

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



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]

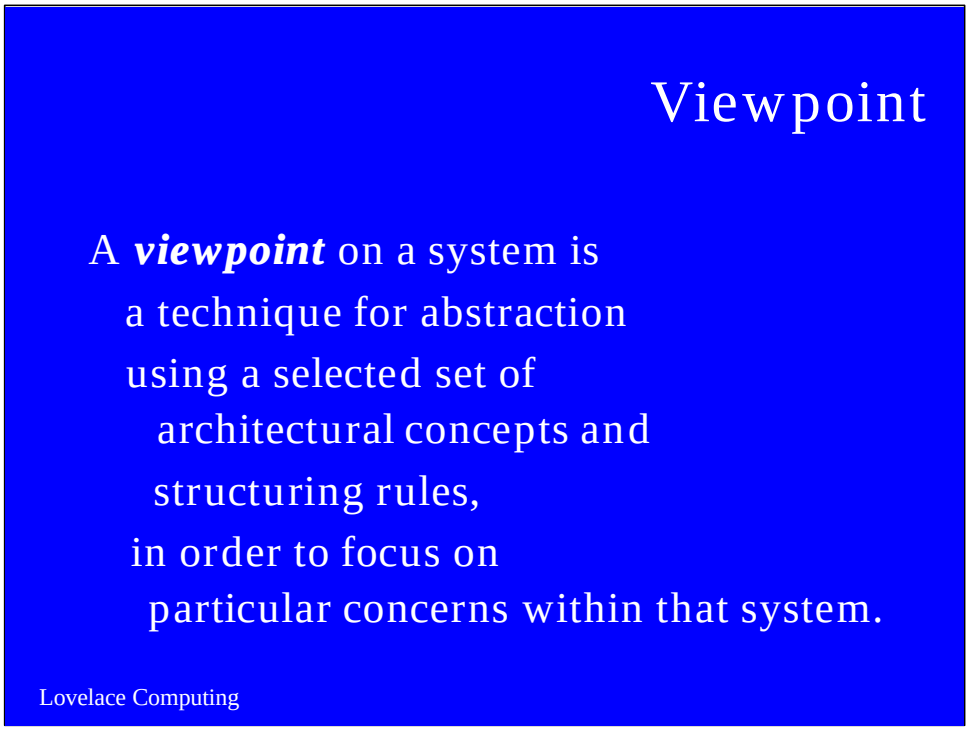
The Model-Driven Architecture prescribes certain kinds of models to be used, how those models may be prepared and the relationships of the different kinds of models.

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.

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

[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



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:

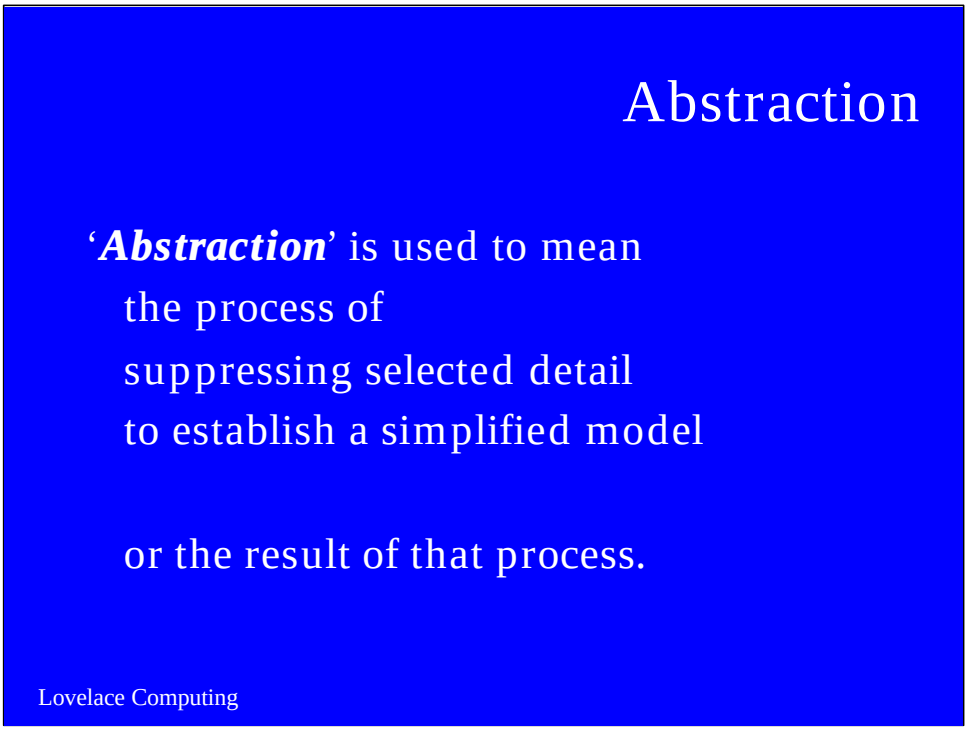
The Reference Model of Open Distributed Processing (ODP) provides five viewpoints for specifying a distributed system. [1]

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.

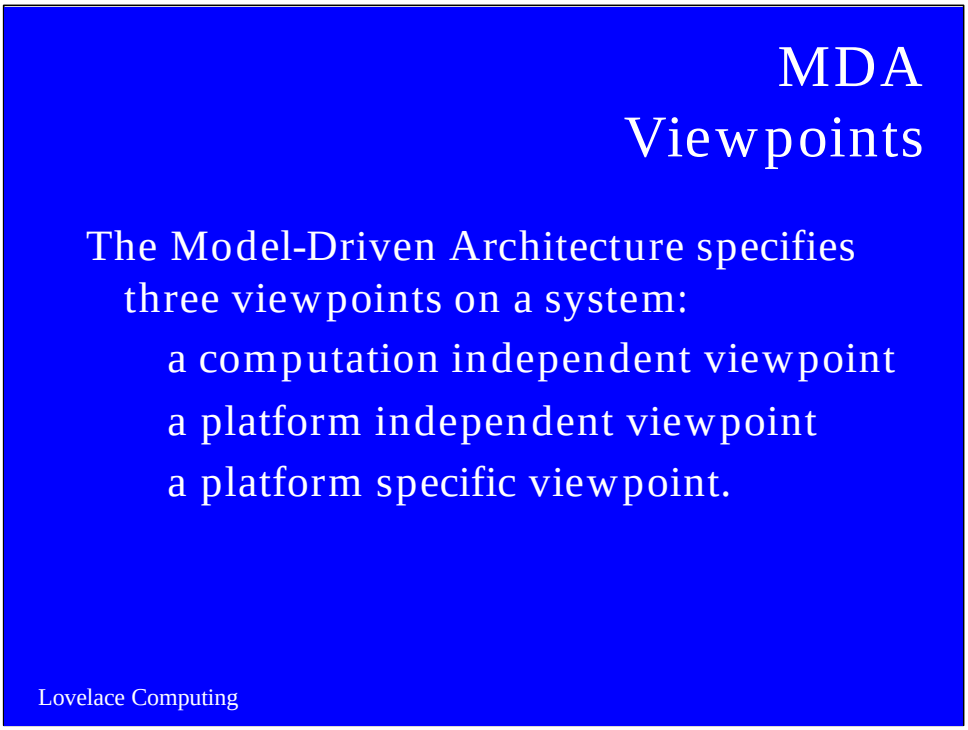
[3] Daniels, Modeling with a Sense of Purpose, IEEE Software, 19:1, January 2002.



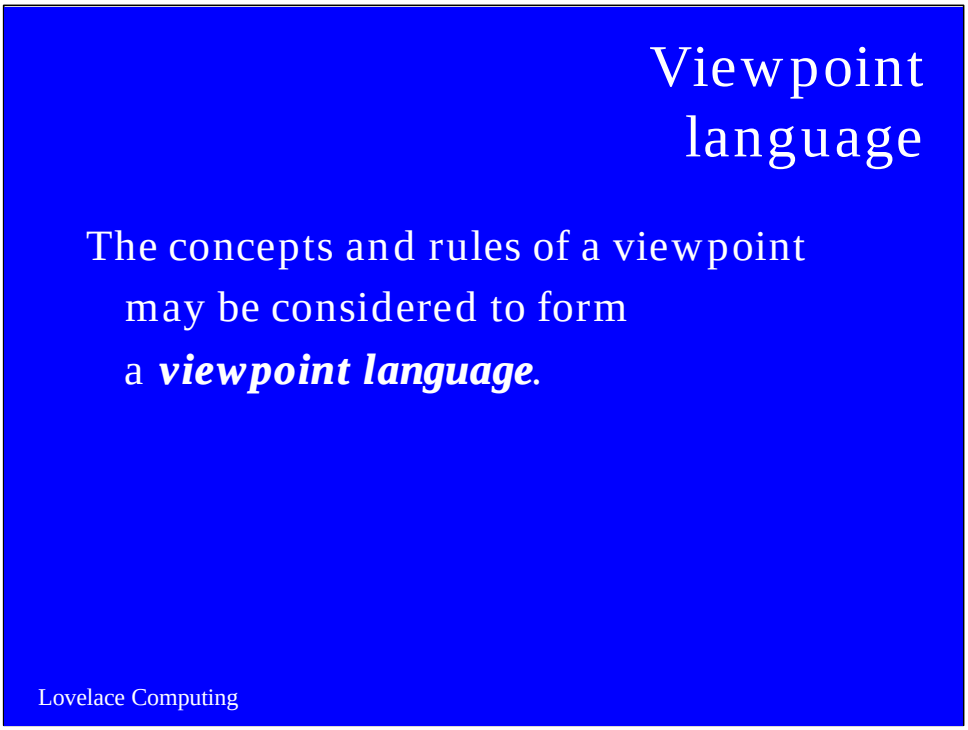
I'll define how I am using 'abstraction,' just so we can be more exact.

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]



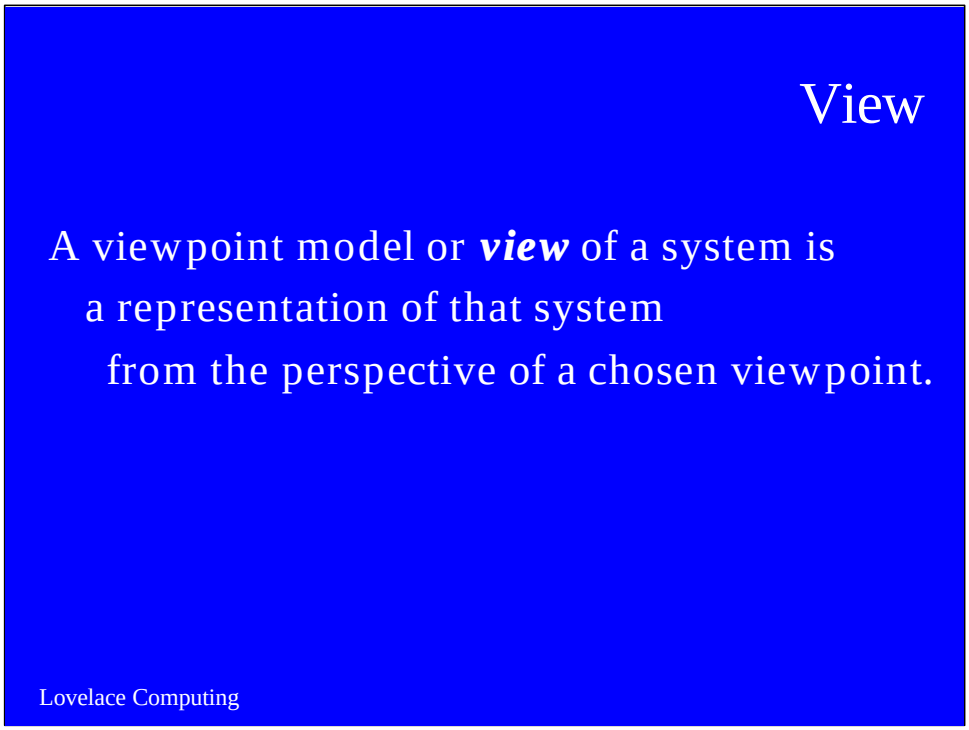
We'll discuss these viewpoints in what follows.



<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.

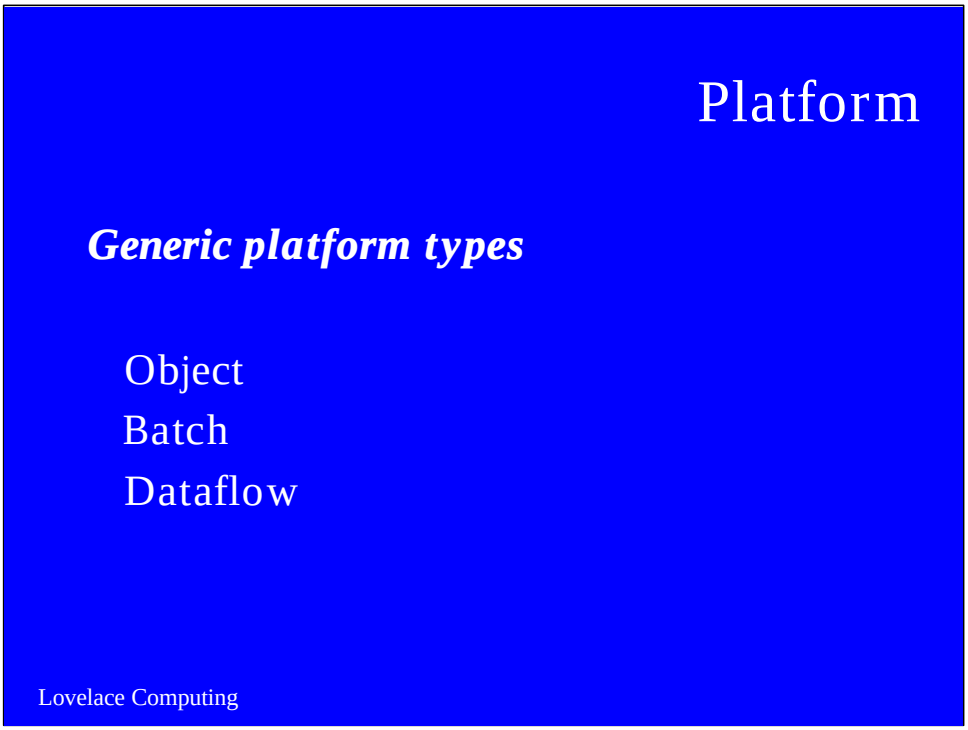
[RM-ODP Part3-4.2.1.1 www.joaquin.net/ODP/Part3/4.html#4.2.1.1]

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:

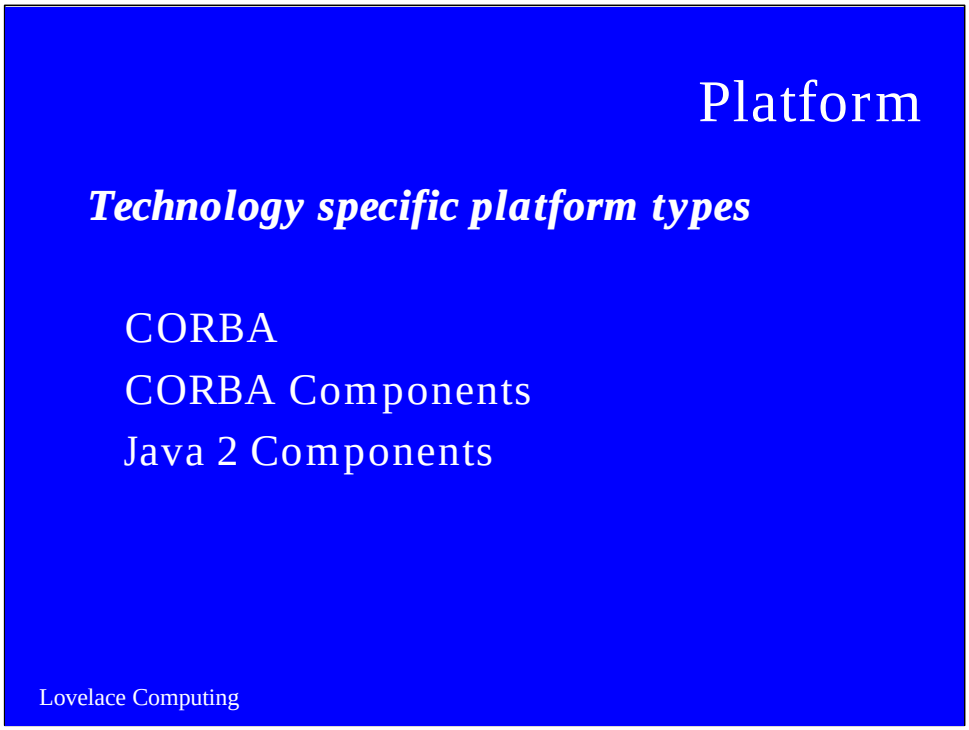
Generic platform types

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

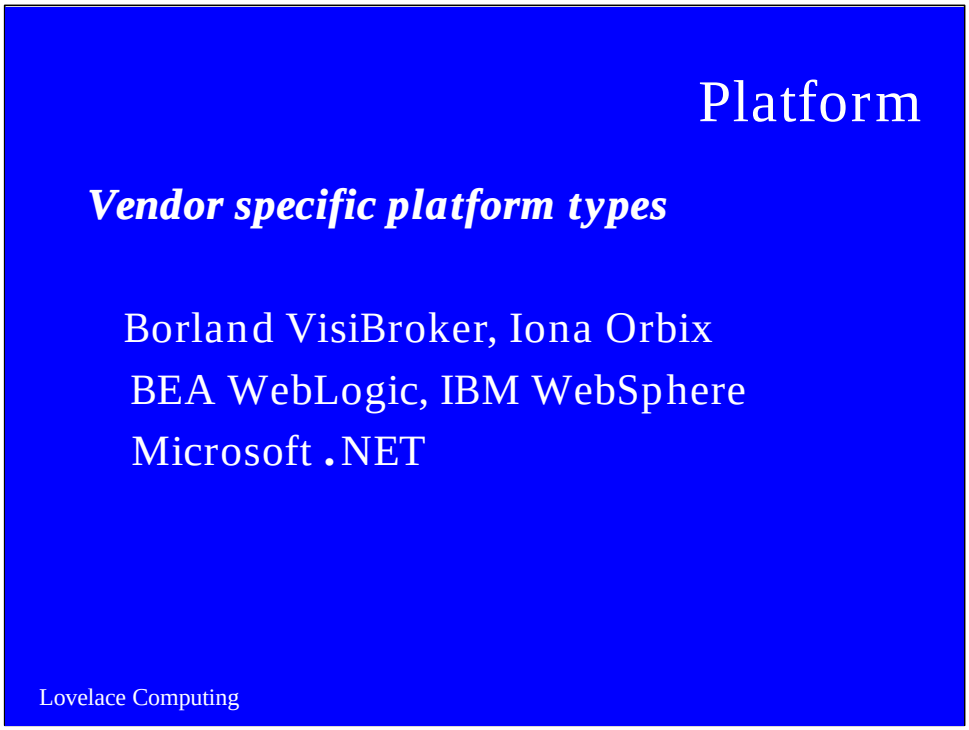


Technology specific platform types

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.



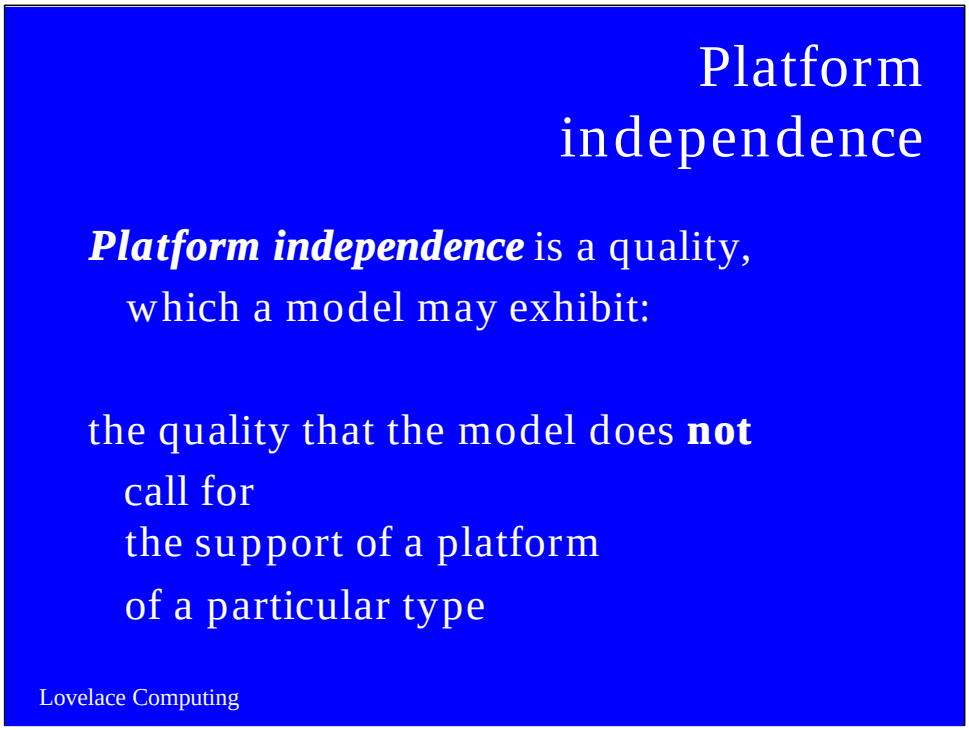
Vendor specific platform types

CORBA: Iona Orbix, Borland VisiBroker, and many others

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

Microsoft .NET

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



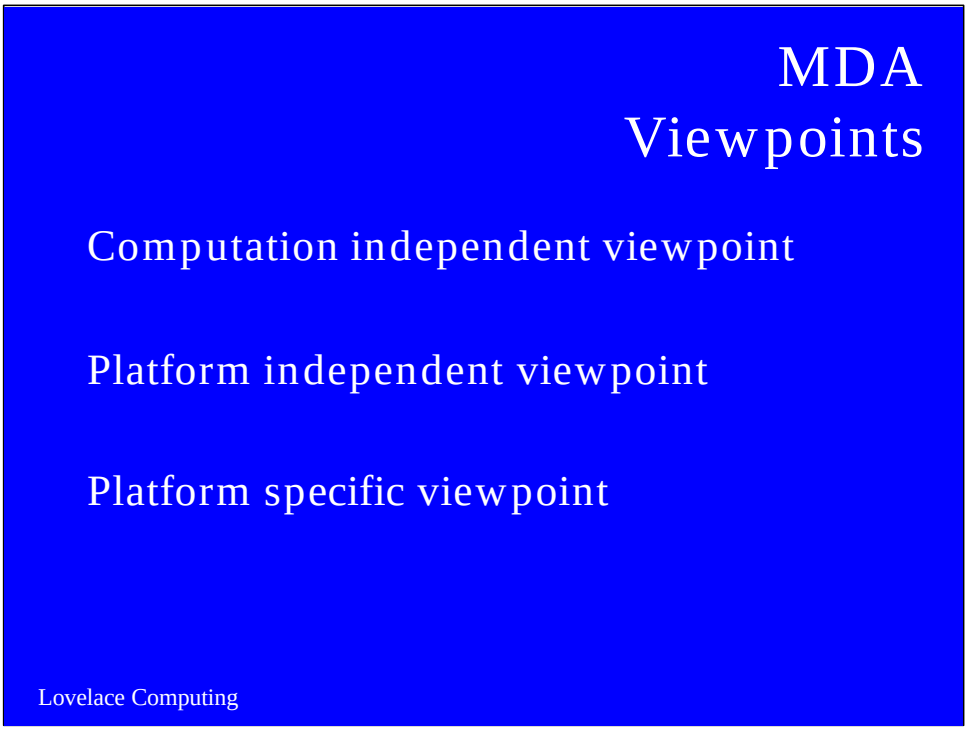
Platform independence is a quality, which a model may exhibit. This is the quality that the model is independent of the features of a platform of a particular type.

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?

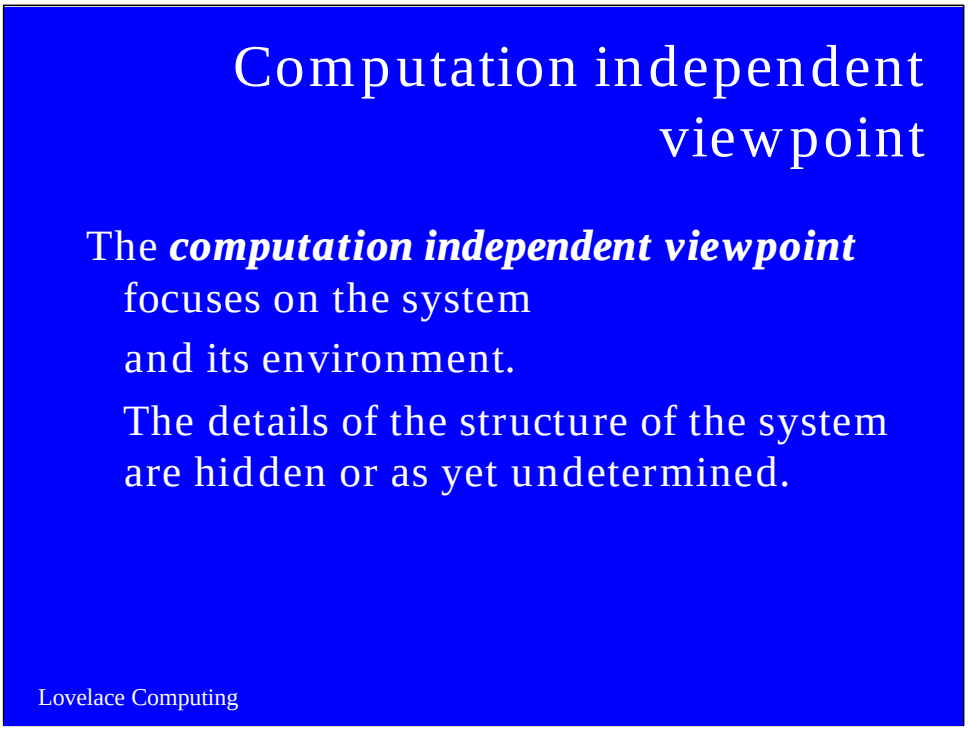
Does anyone have a different take on the concepts discussed so far?

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.



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.

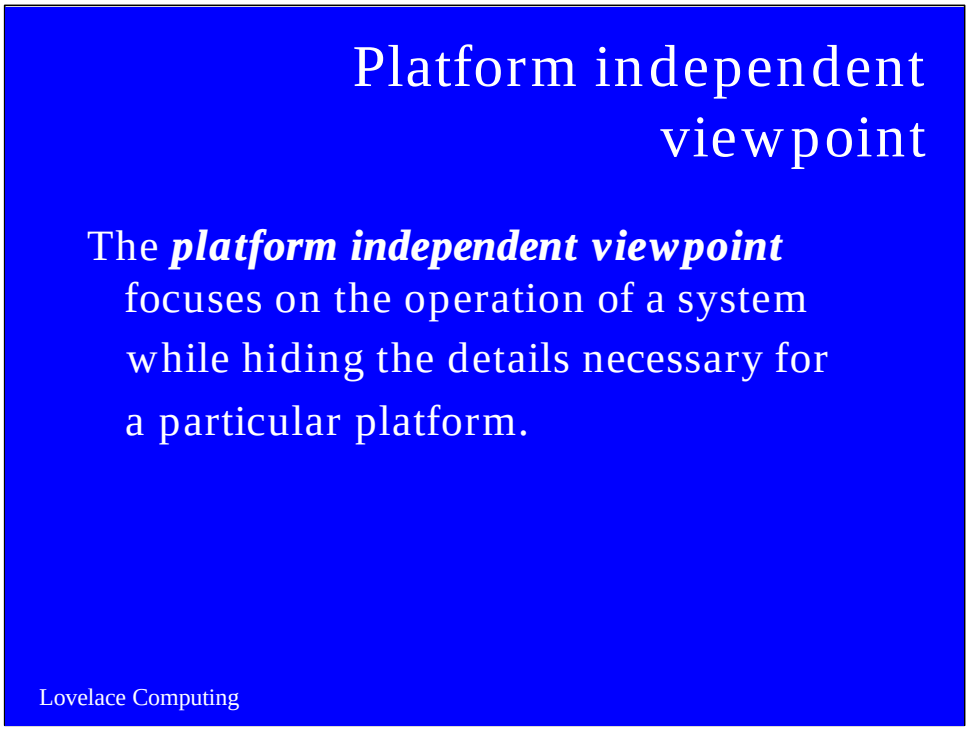
It is assumed that the primary user of the CIM, the domain practitioner, is not knowledgeable about the models or artifacts used to realize the functionality for which the requirements are articulated in the CIM. The CIM plays an important role in bridging the gap between those that are experts about 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]

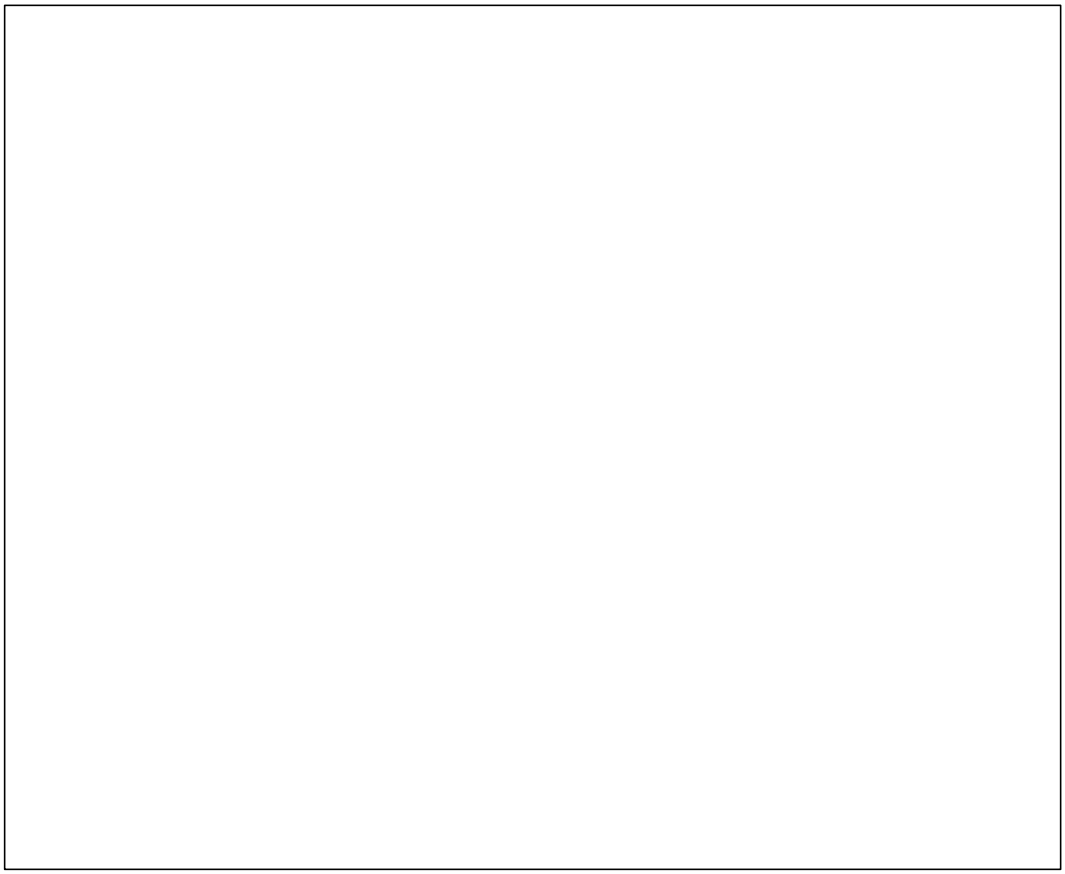


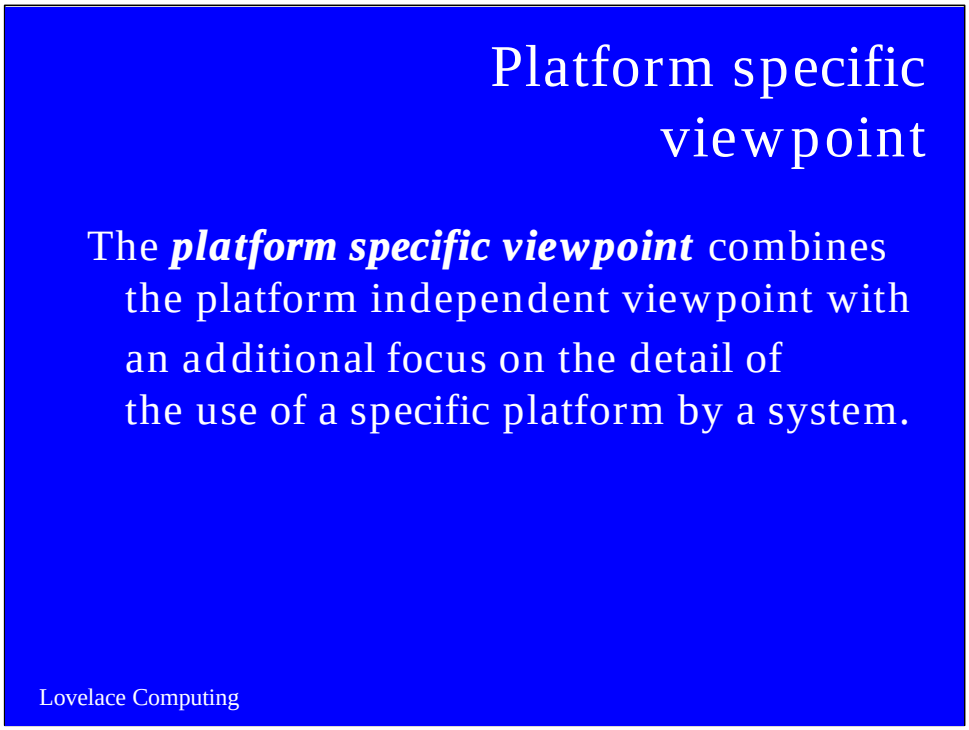
The *platform independent viewpoint* focuses on the operation of a system while hiding the details necessary for a particular platform. A platform independent view shows that part of the complete specification that does not change from one platform to another.

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.





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)

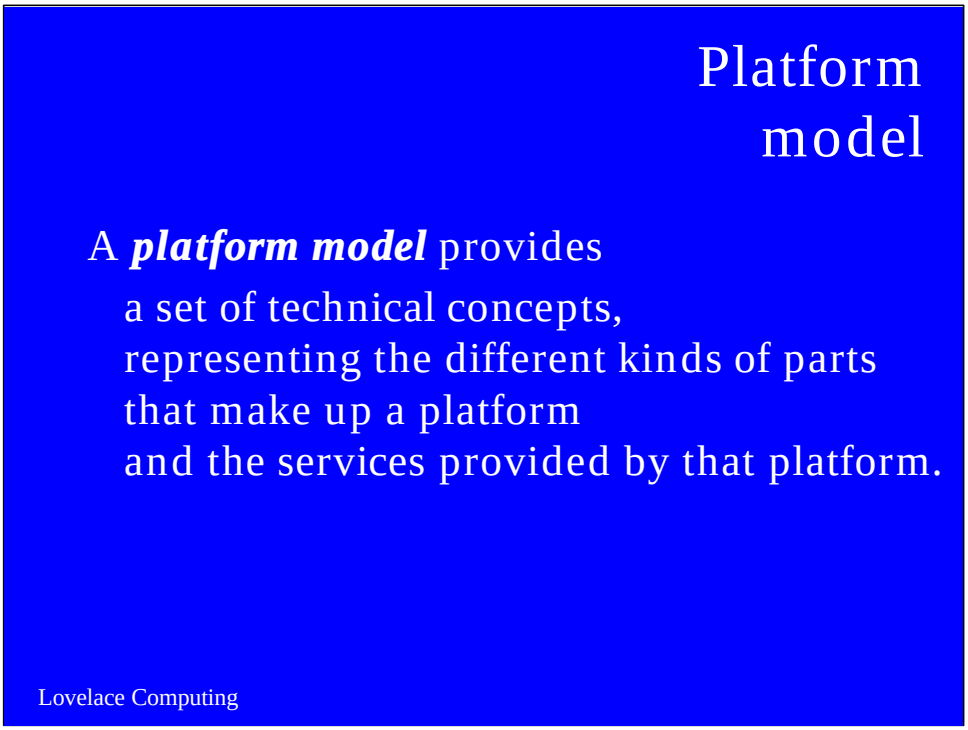
A *platform independent model* is a view of a system from the platform independent viewpoint.

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

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.



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.

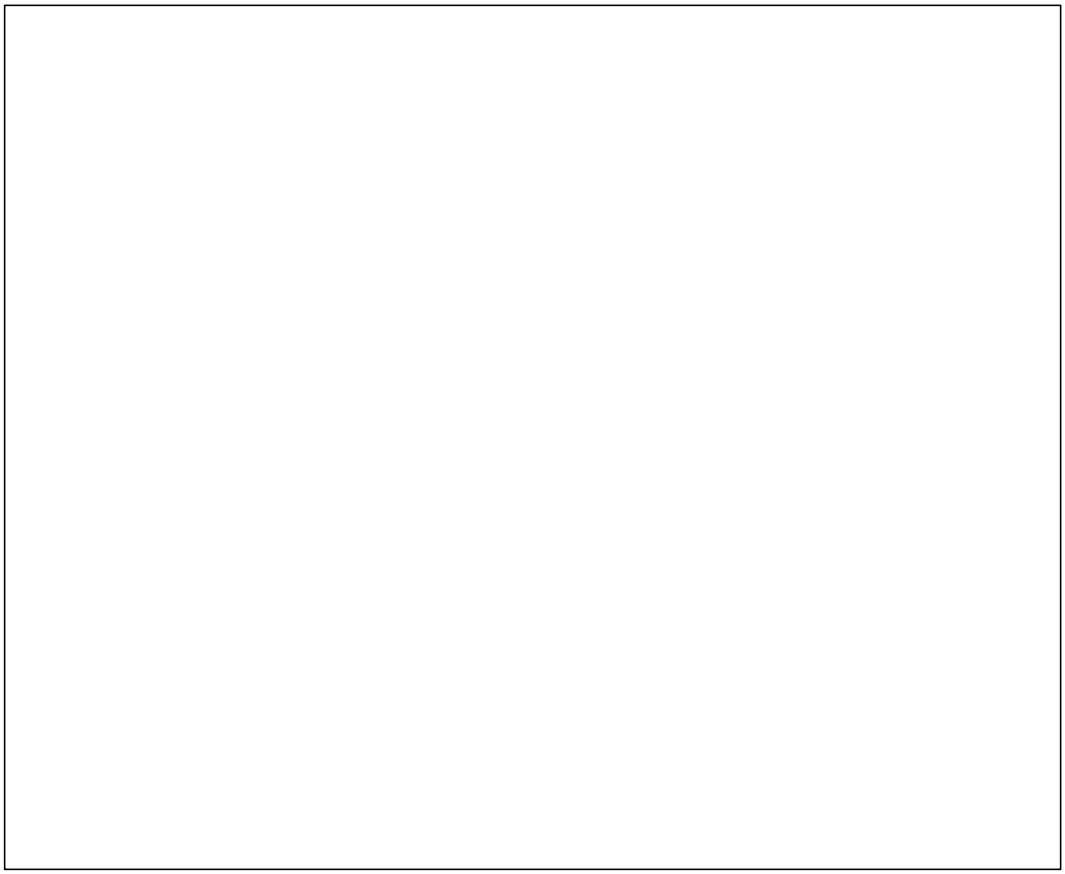
A platform model also specifies requirements on the connection and use of the parts of the platform, and the connections of an application to the platform.

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.



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.

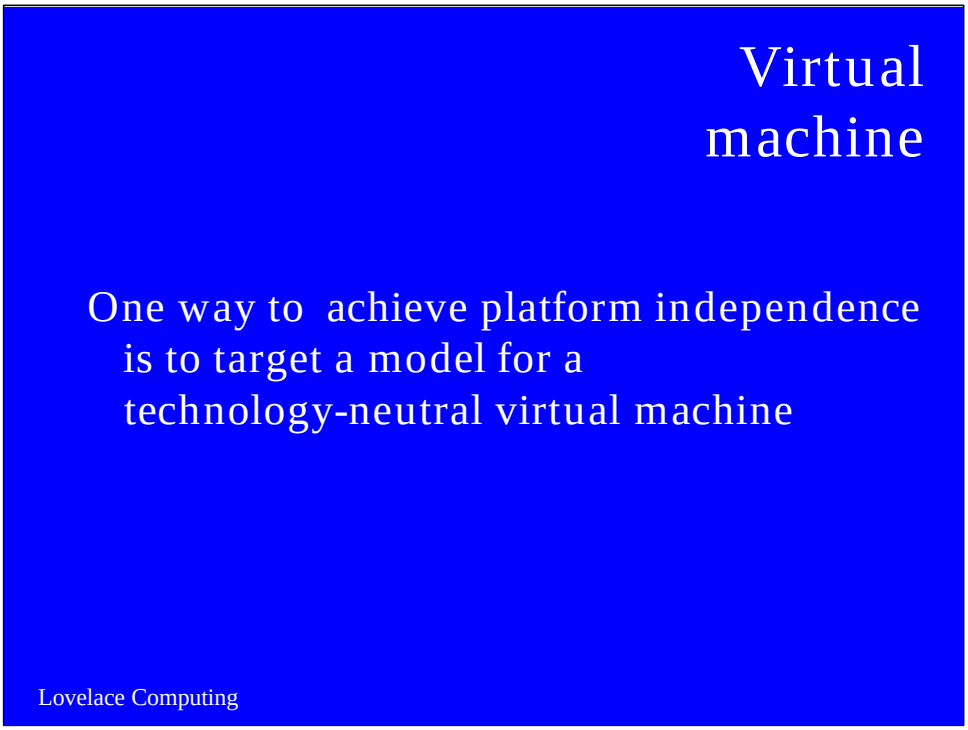
Platform model

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.



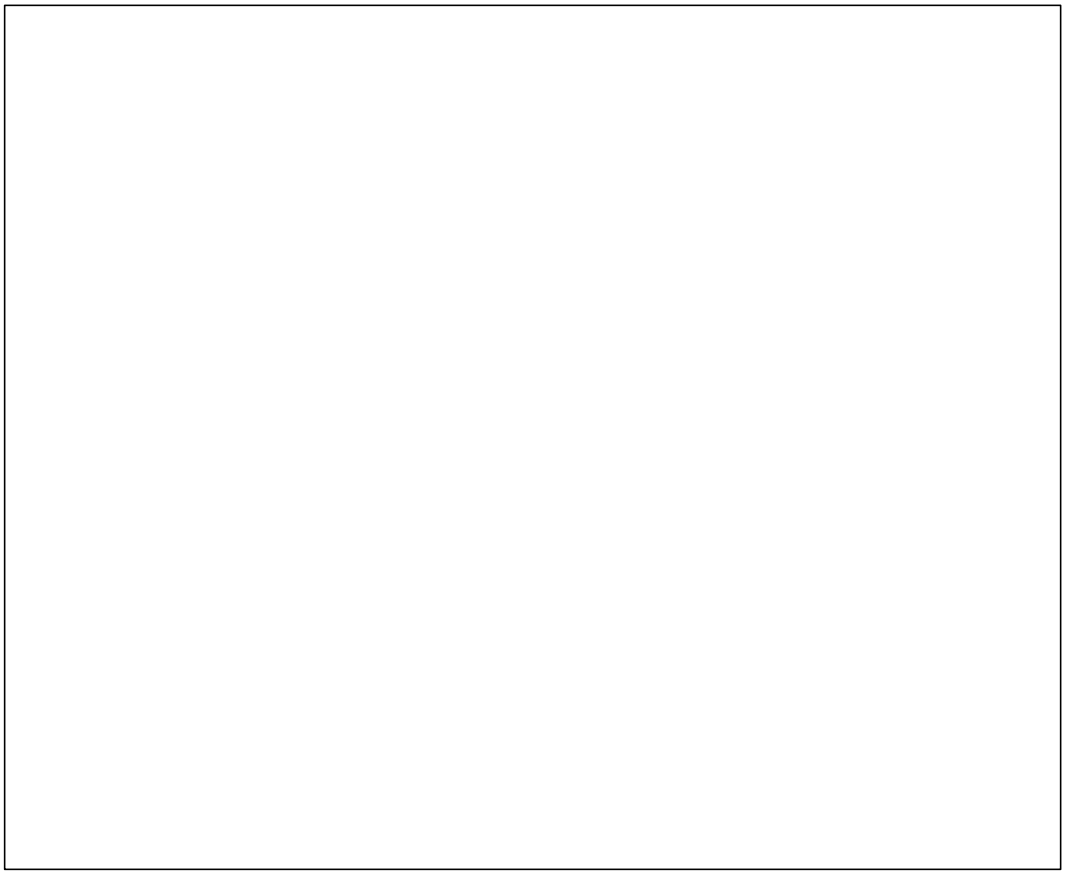
A very common technique for achieving platform independence is to target a system model for a technology-neutral virtual machine. A virtual machine is a system that is specified independently of any specific platform and which is realized in platform-specific ways on different platforms. A virtual machine 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**.

Questions?

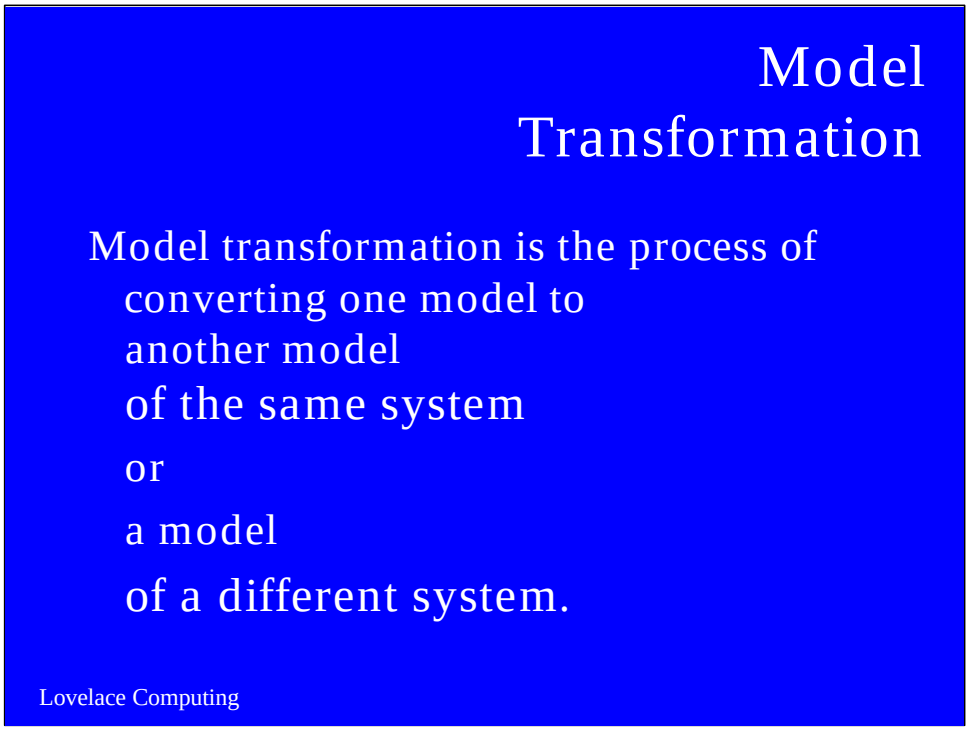
Does anyone have a different take on the concepts discussed so far?

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.

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.

[ ] 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

Lovelace Computing

How MDA is used

PIM

Mapping

Mark

Transformation

Record

PSM

Lovelace Computing

Model

Domain model

Computation independent model

Platform independent model

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

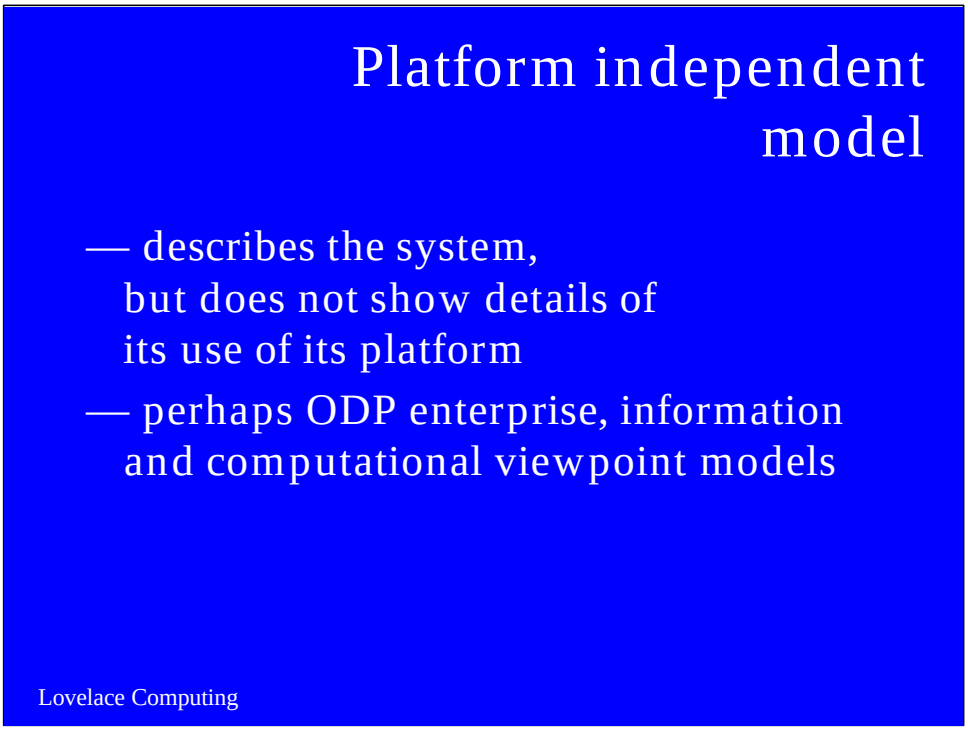
— independent of how the system is implemented.

—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.

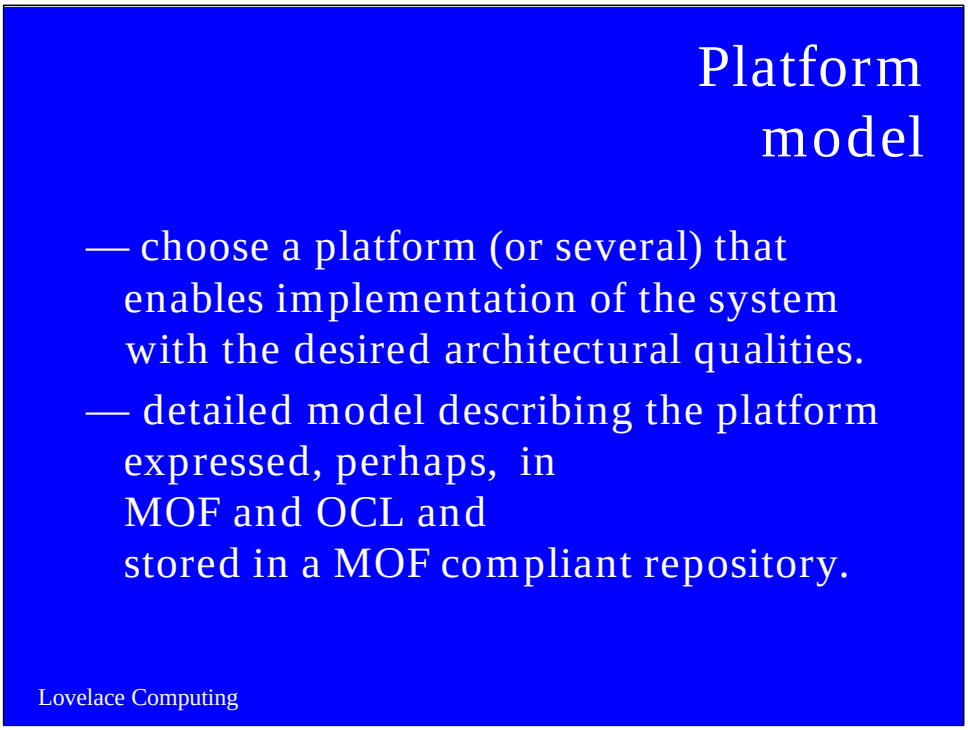
A computation independent model might consist of two UML models, from the ODP enterprise and information viewpoints. It might include several models from these viewpoints, some providing more detail than others, or focusing on particular concerns of a viewpoint.



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

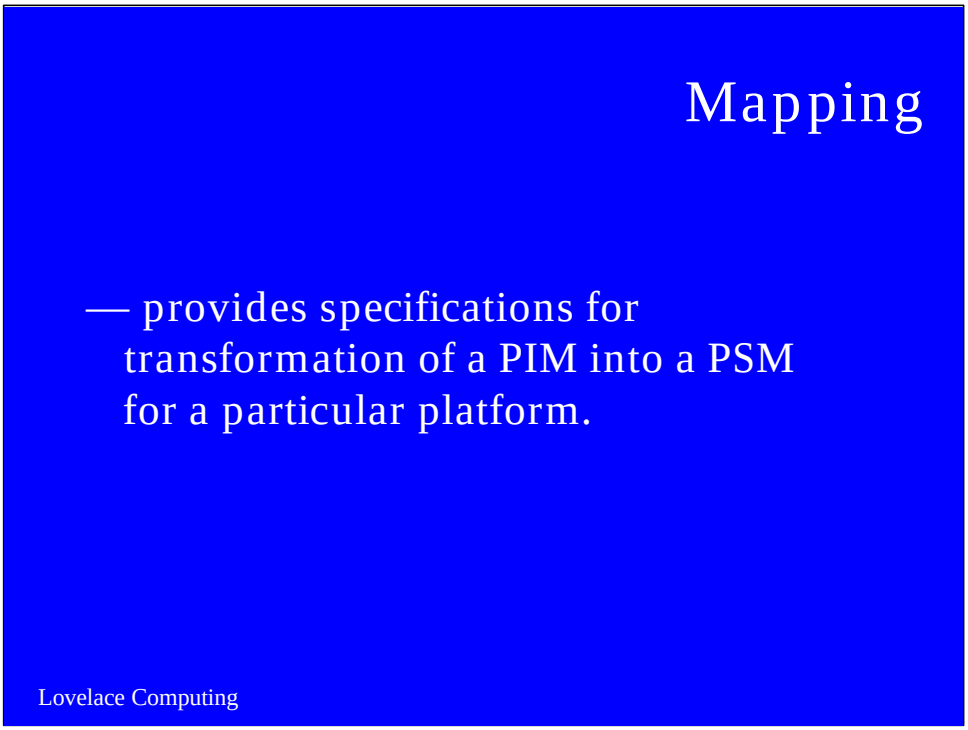
Mark

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.

How MDA is used

PIM

Mapping

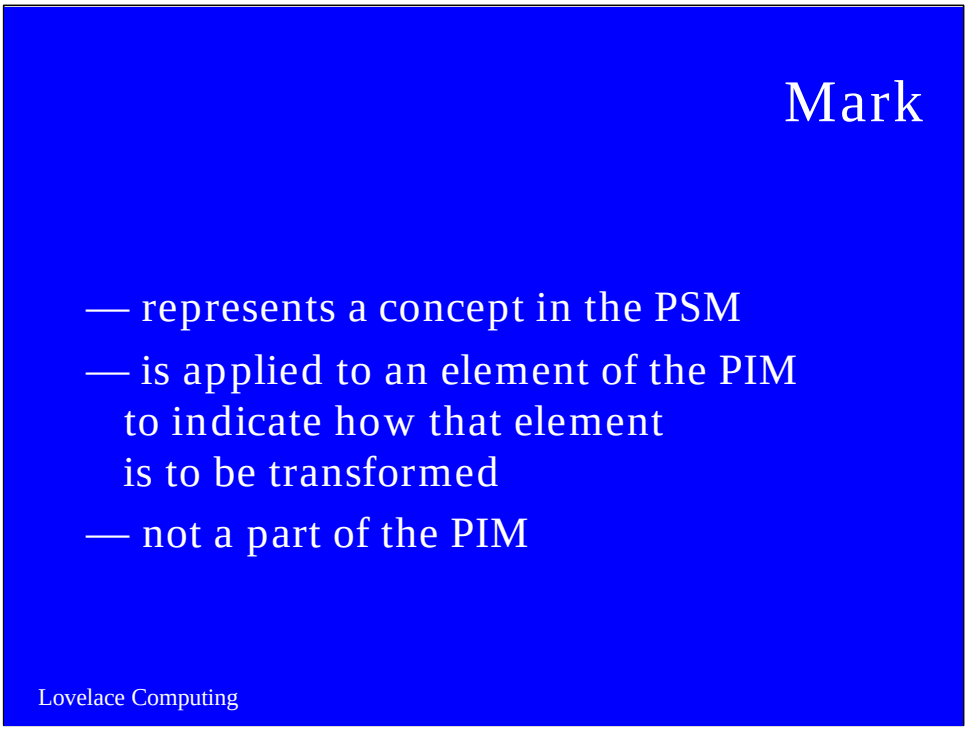
Mark

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.

How MDA is used

PIM

Mapping

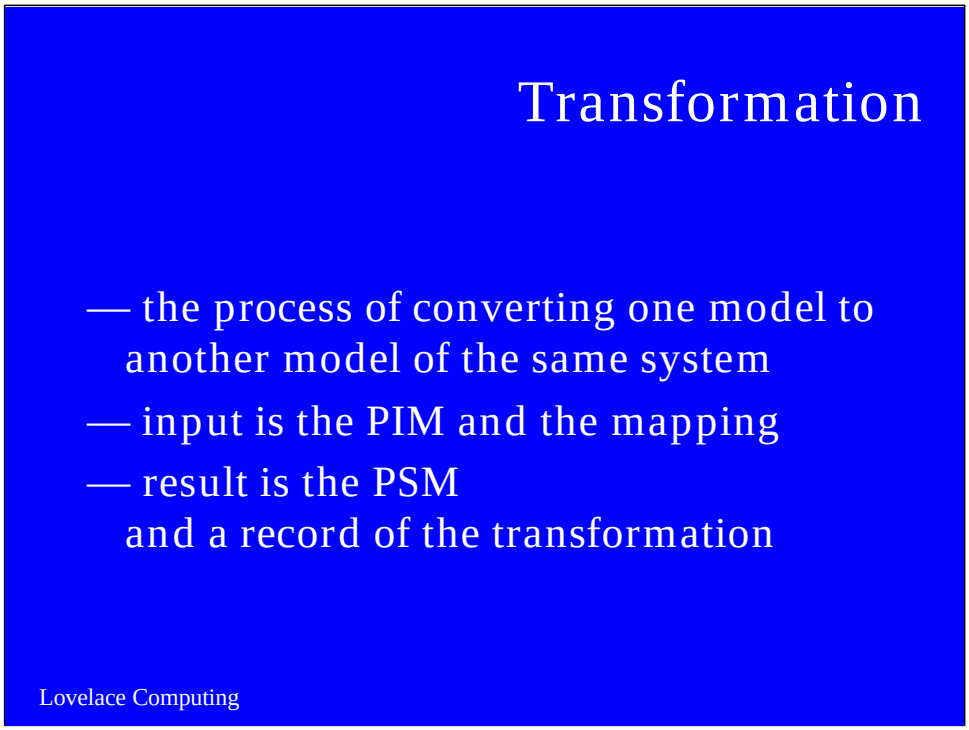
Mark

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.

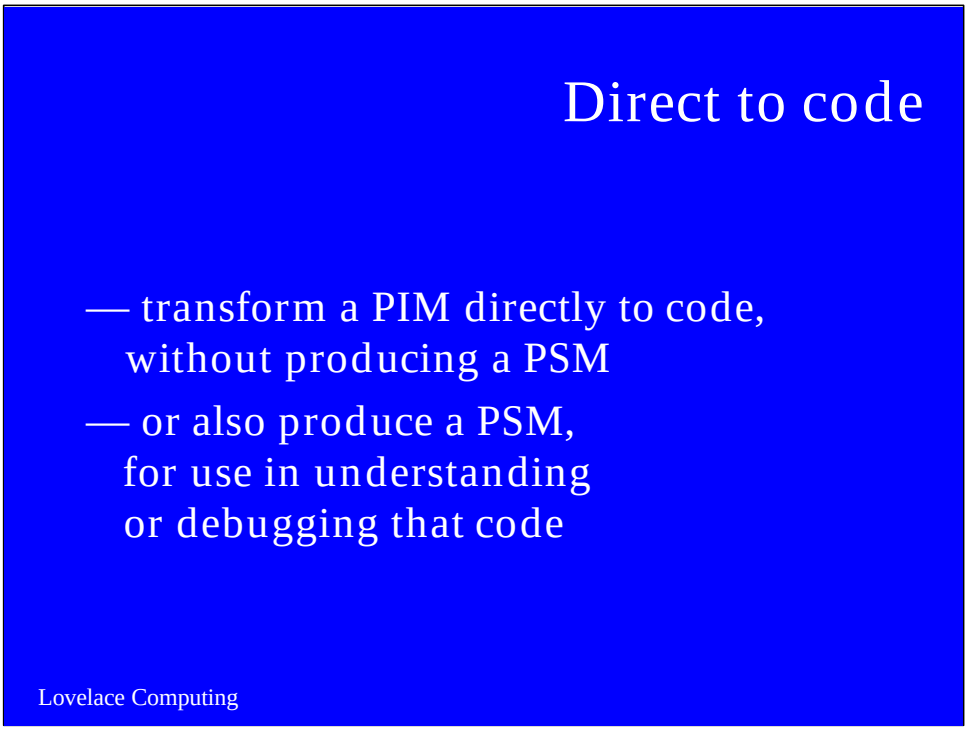
Model transformation is the process of converting one model to another model of the same system. The input to the transformation is the marked PIM and the mapping. The result is the PSM and the record of transformation.

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.



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

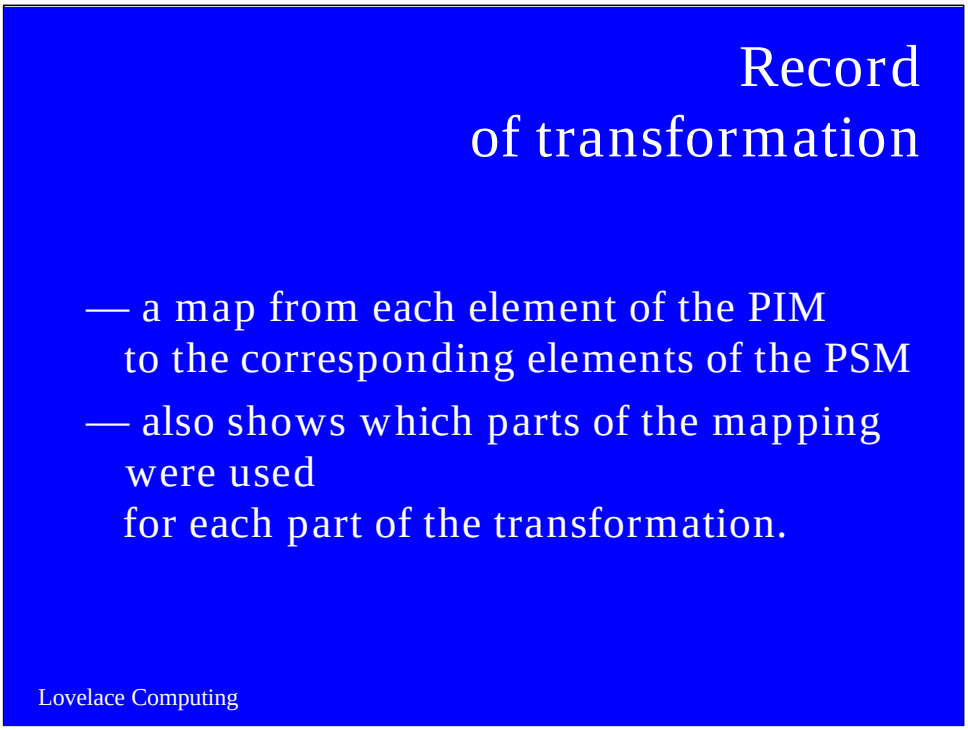
Mark

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.

How MDA is used

PIM

Mapping

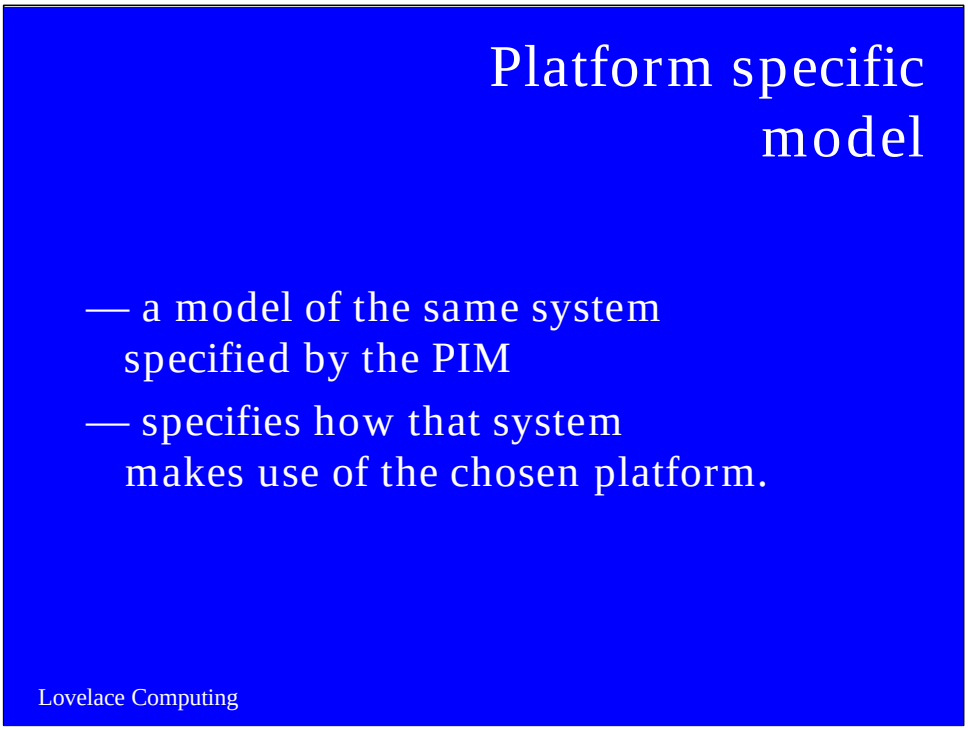
Mark

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.

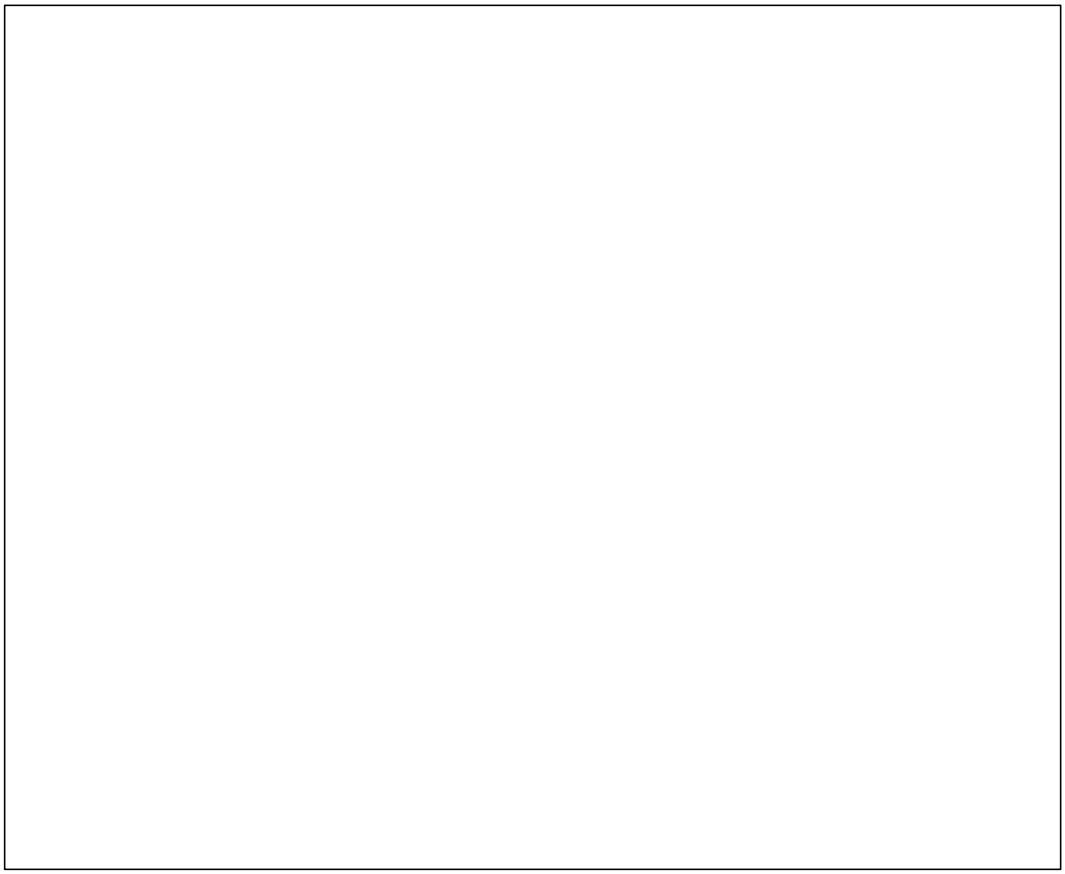
A PSM may provide more or less detail, depending on its purpose. A PSM will be an implementation, if it provides all the information needed to construct a system and to put it into operation.

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.

Lovelace Computing



Implementation

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

Lovelace Computing

How MDA is used

PIM

Mapping

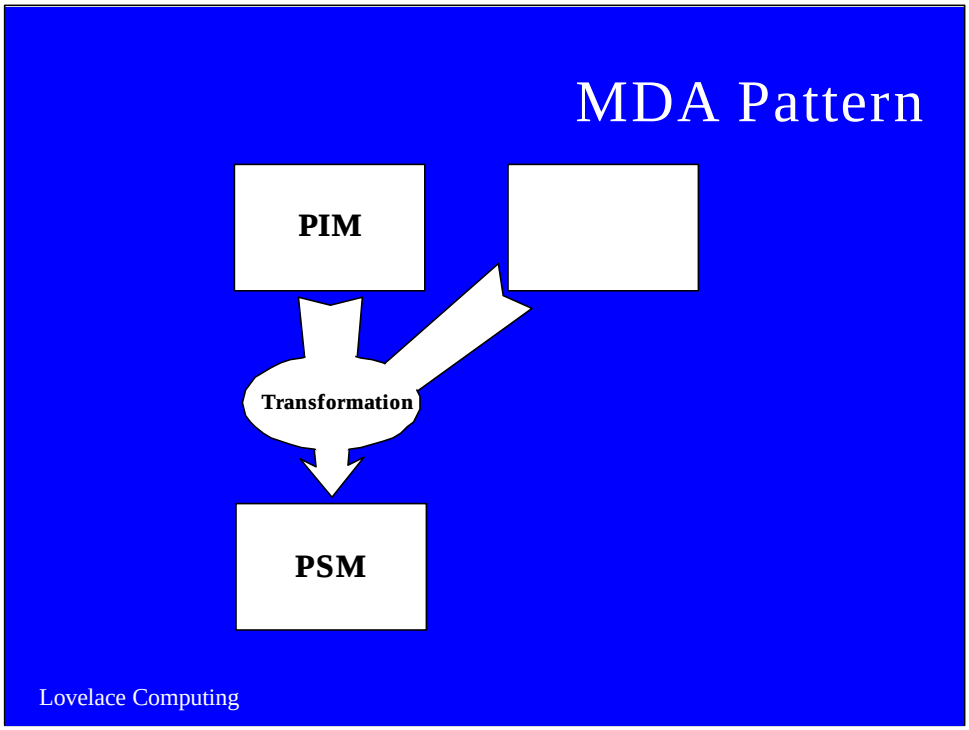
Mark

Transformation

Record

PSM

Lovelace Computing

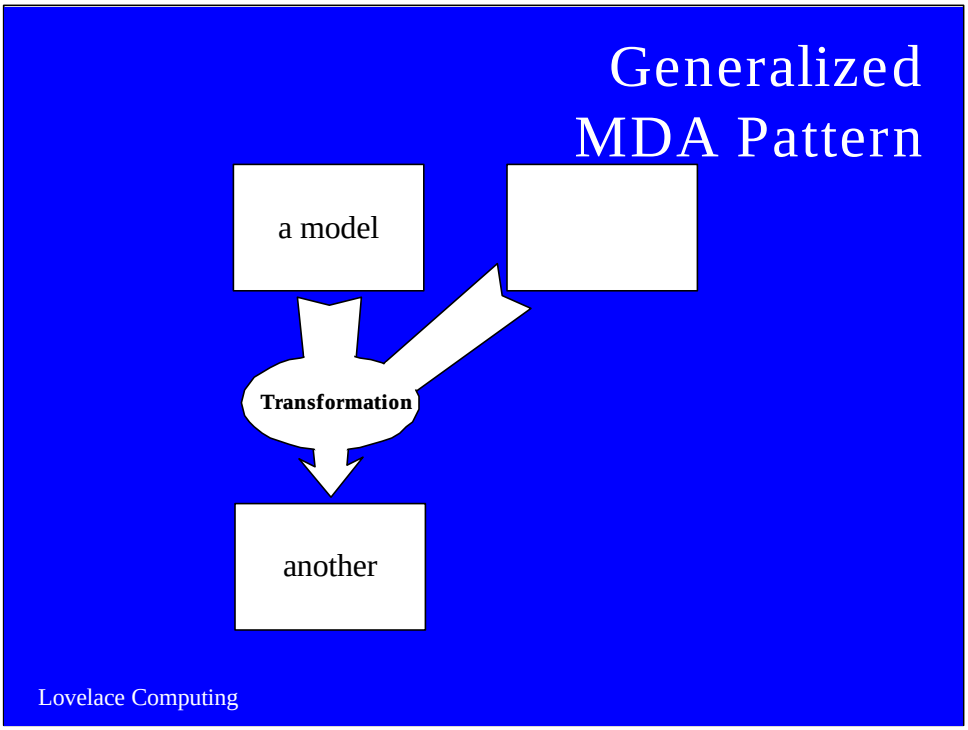


The drawing illustrates the MDA pattern, by which a PIM is transformed to a PSM.

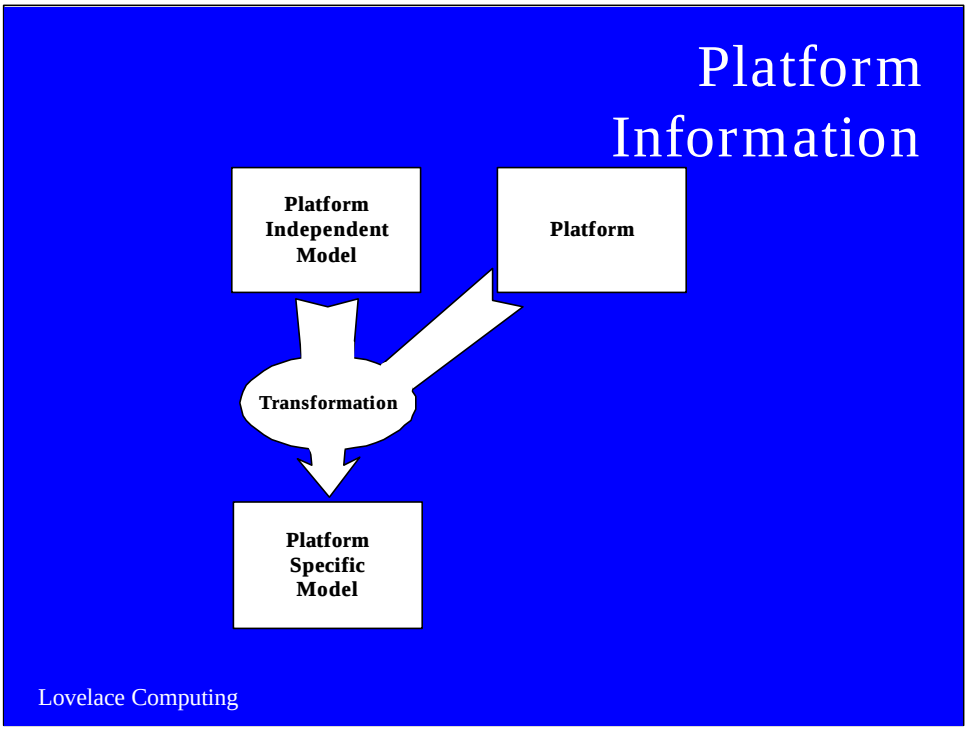
The drawing is intended to be suggestive. The platform independent model and other information are combined by the transformation to produce a platform specific model.

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.

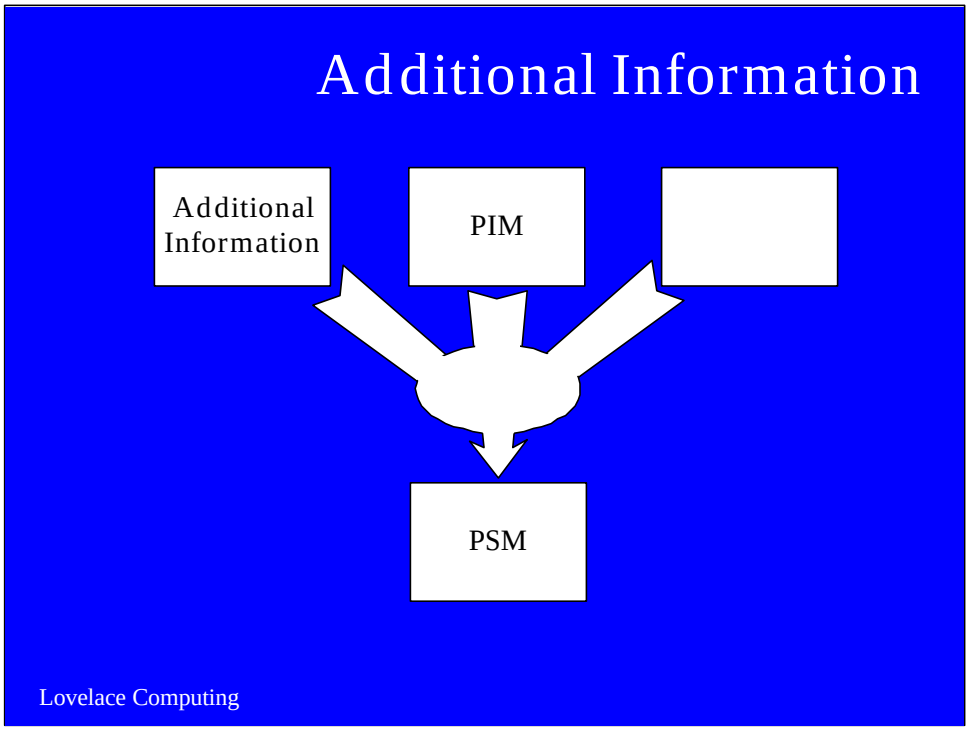
The box with no name represents what goes into the transformation, in addition to the platform independent model. This varies with different styles of MDA.



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.

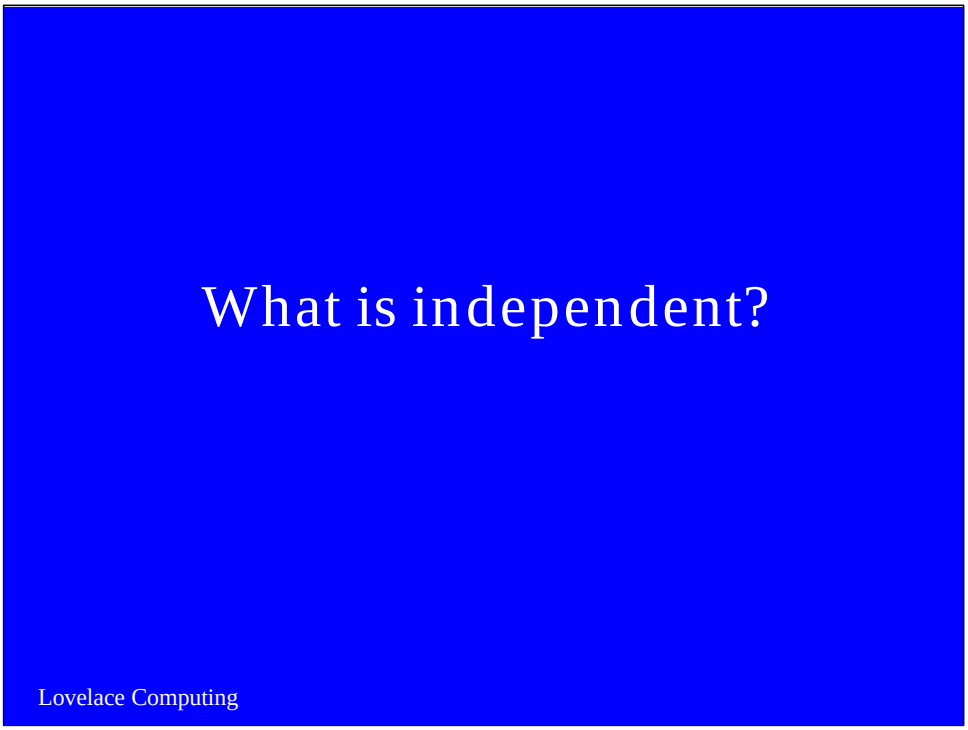
In addition to the PIM and the platform information, additional information can be supplied to guide the transformation.

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.

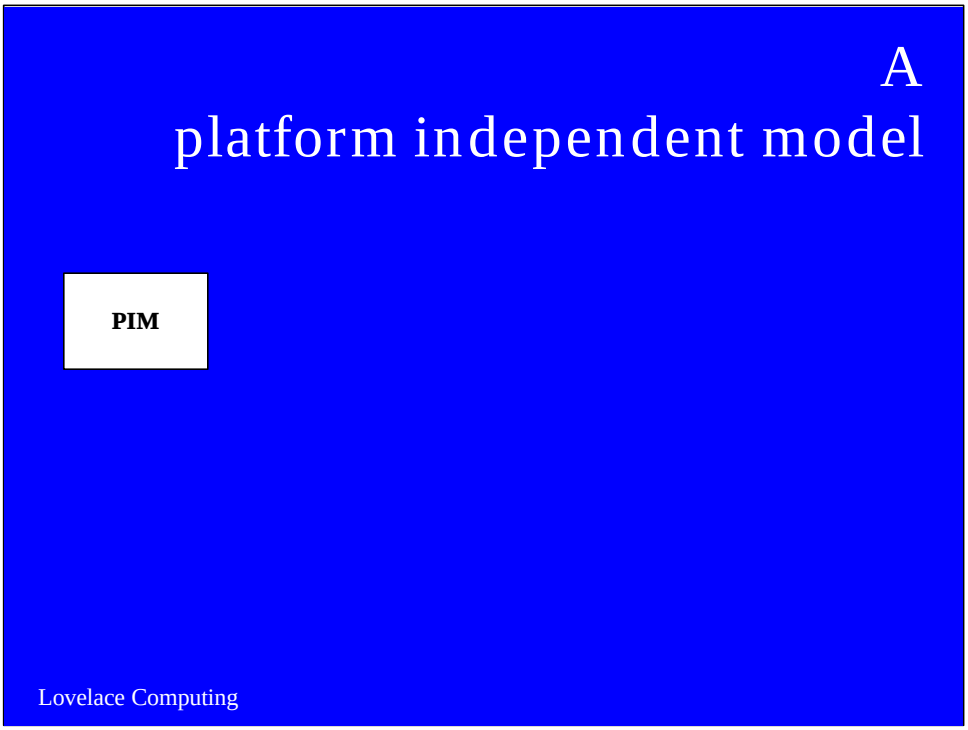
Often the additional information will draw on the practical knowledge of the designer. This will be both knowledge of the application domain and knowledge of the platform.

Using the MDA pattern

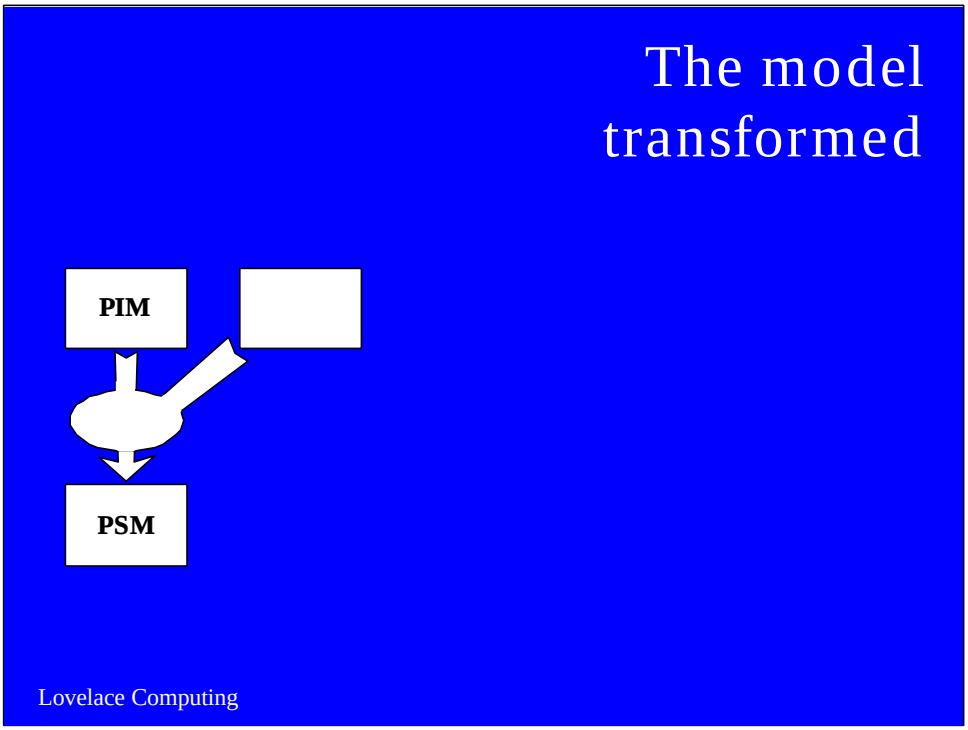
Lovelace Computing



The MDA pattern may be applied more than once.

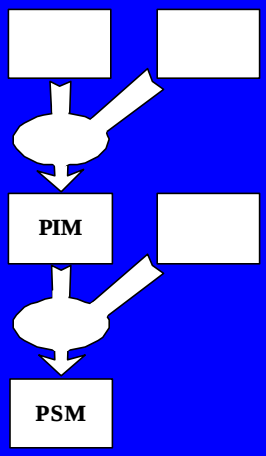


The original PIM is an application model, designed to be independent of many platform choices.



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 model transformed again

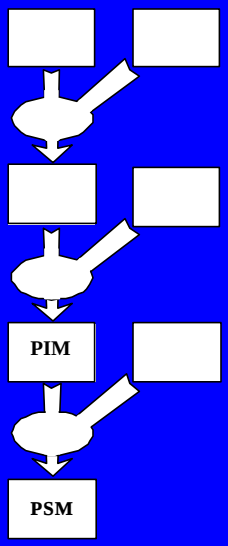


Lovelace Computing

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.

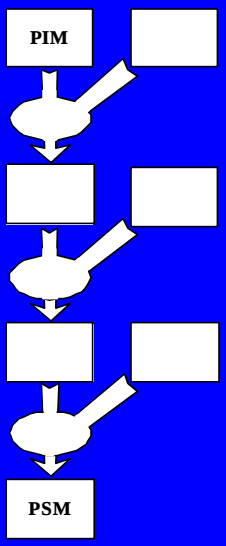
The model transformed a third time



Lovelace Computing

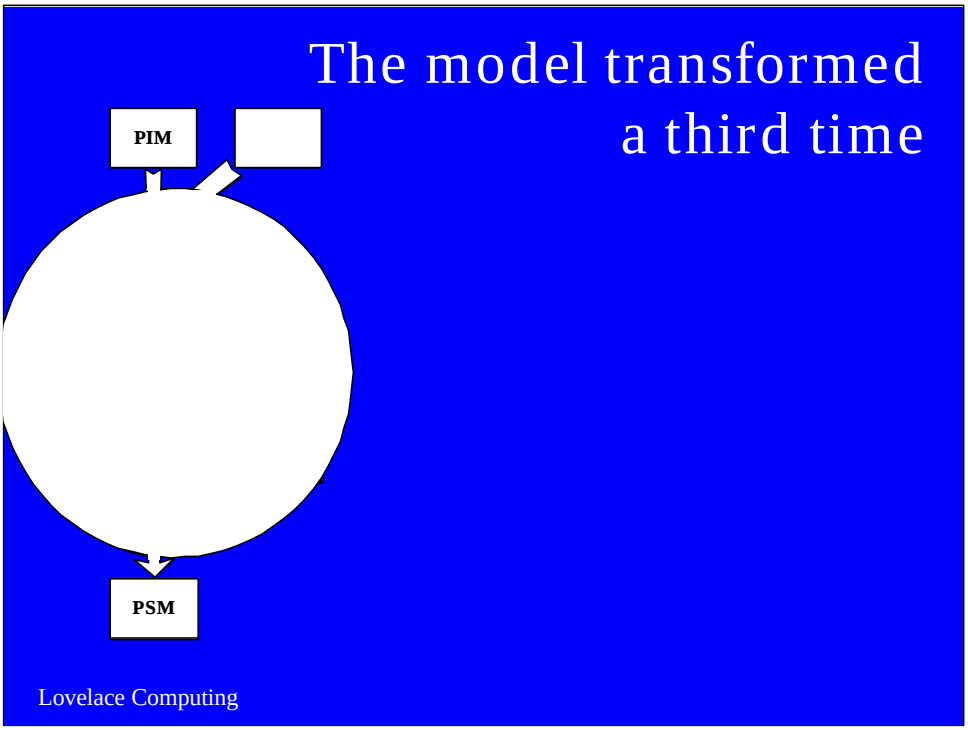
It may be desirable to transform this model again, to make specialized use of the platform in order to achieve a certain quality of service, perhaps to meet an availability requirement.

The model transformed a third time



Lovelace Computing

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

What is a platform?

Lovelace Computing

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

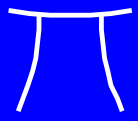
Lovelace Computing

Some notation

An application

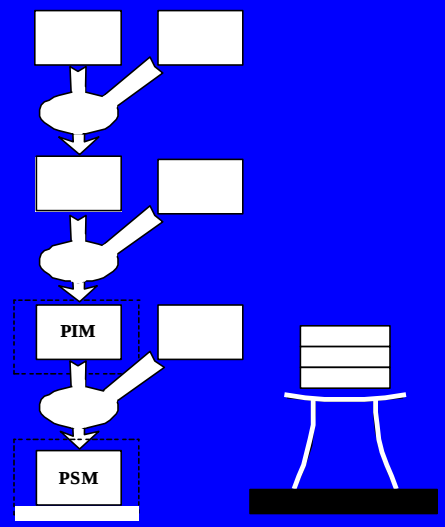
A system consists of one or more ***applications***, supported by one or more platforms

Some notation



A platform

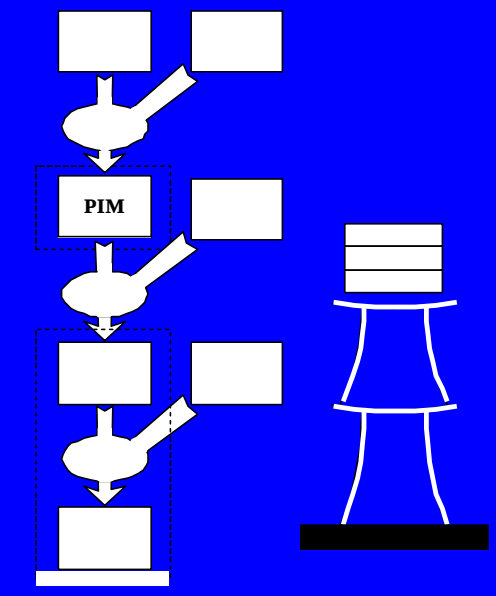
A platform specific model



Lovelace Computing

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.

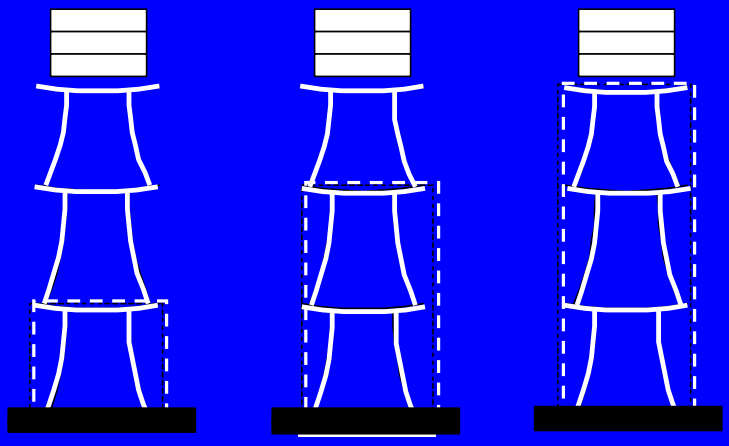
Another platform specific model



Lovelace Computing

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

What counts as a platform?



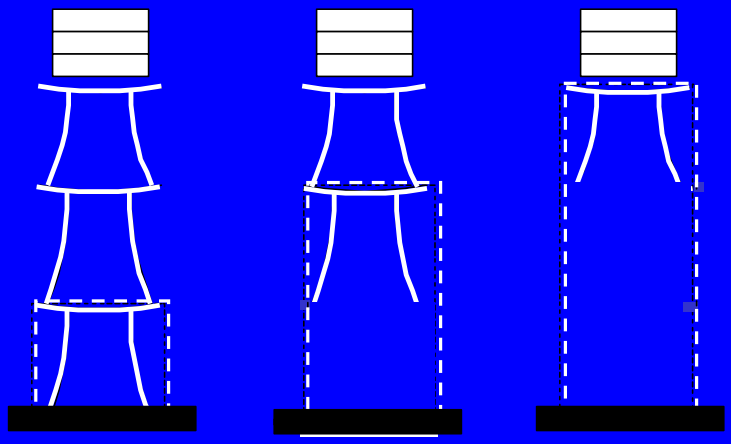
Lovelace Computing

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.

Part of a platform hidden by abstraction

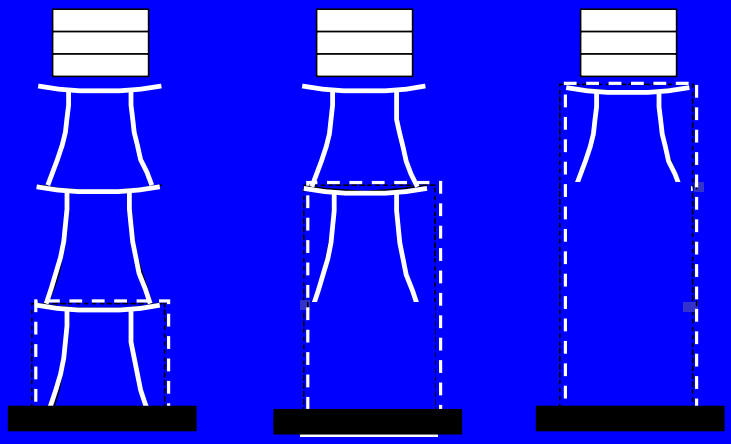


Lovelace Computing

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.

Part of a platform hidden by abstraction



Lovelace Computing

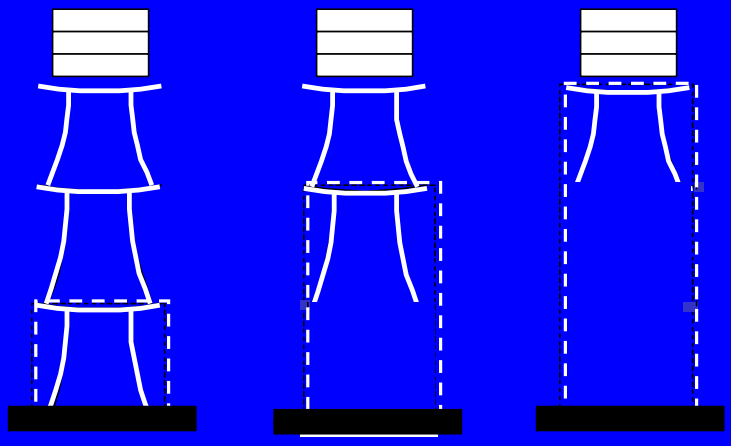
When a platform provides a degree of portability, it is appropriate to hide the details of the particular supporting platform, since portability makes it possible to choose one or another supporting platform.

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).

Part of a platform hidden by abstraction



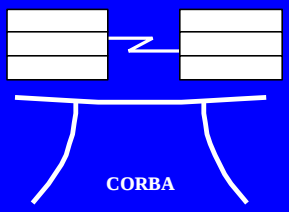
Lovelace Computing

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.

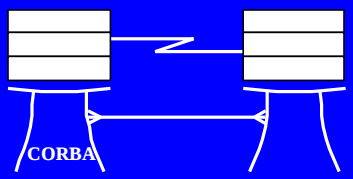
What counts as a platform is relative to the purpose of the modeler. For many MDA users, middleware is a platform, for a middleware developer an operating system is the platform. Thus a platform-independent model of middleware might appear to be a highly platform-specific model from the point of view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

Interoperability: a common platform



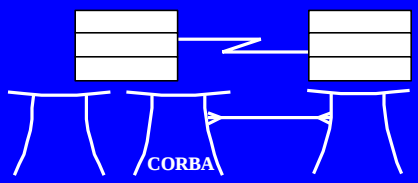
Lovelace Computing

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

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
joaquin.no.spam that-sign acm the-dot org

www.joaquin.net