The several styles of Model Driven Architecture

Joaquin Miller
Lovelace® Computing
representing X-Change Technologies

This presentation discusses the several different styles of Model Driven Architecture.

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/

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)

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.

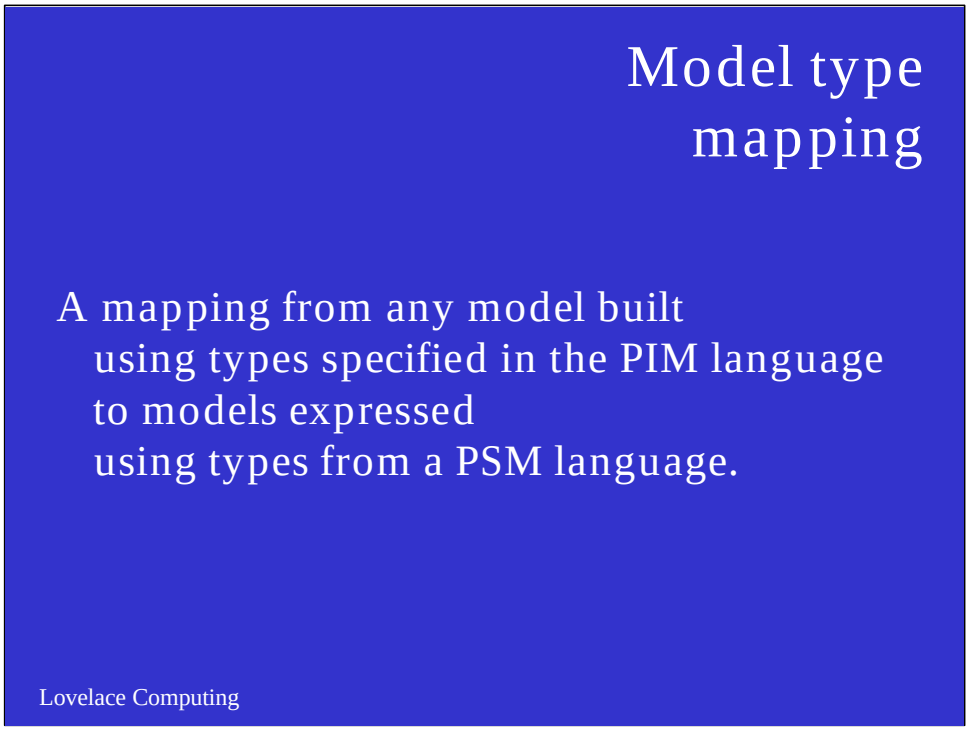
Examples

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.



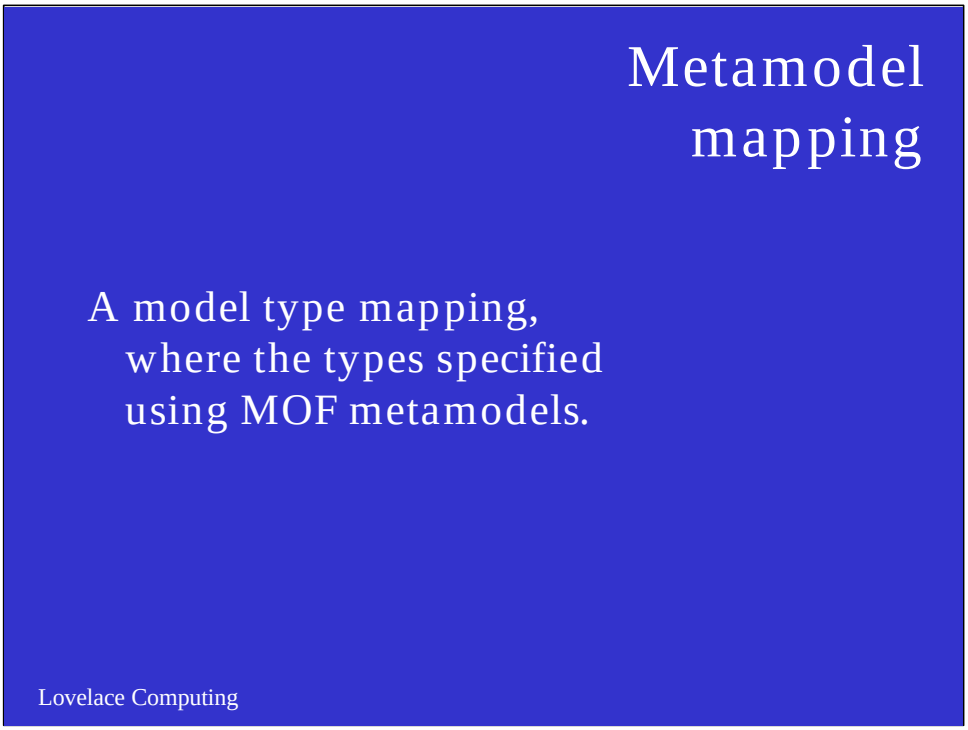
A model type mapping specifies a mapping from any model built using types specified in the PIM language to models expressed using types from a PSM language.

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.

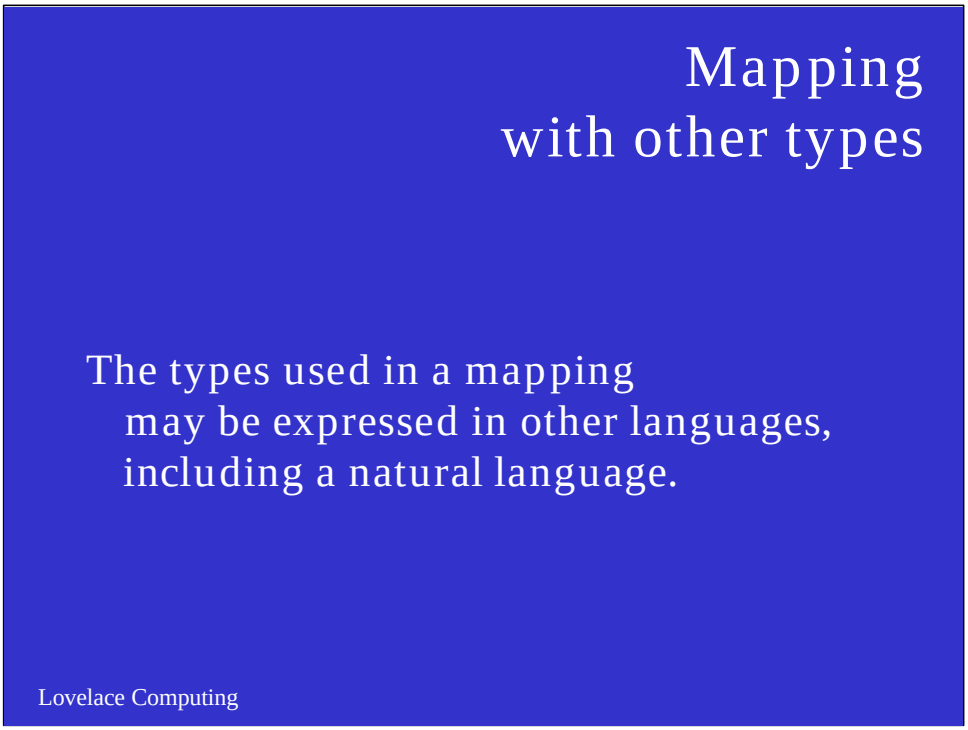
Example: If pattern exists where an instance of class, Entity, has a manages association to an instance of class, Document, whose attribute, persistent, is set, then map that instance to an EJB Entity which manages whatever is mapped from the instance of Document instance identified by the pattern.



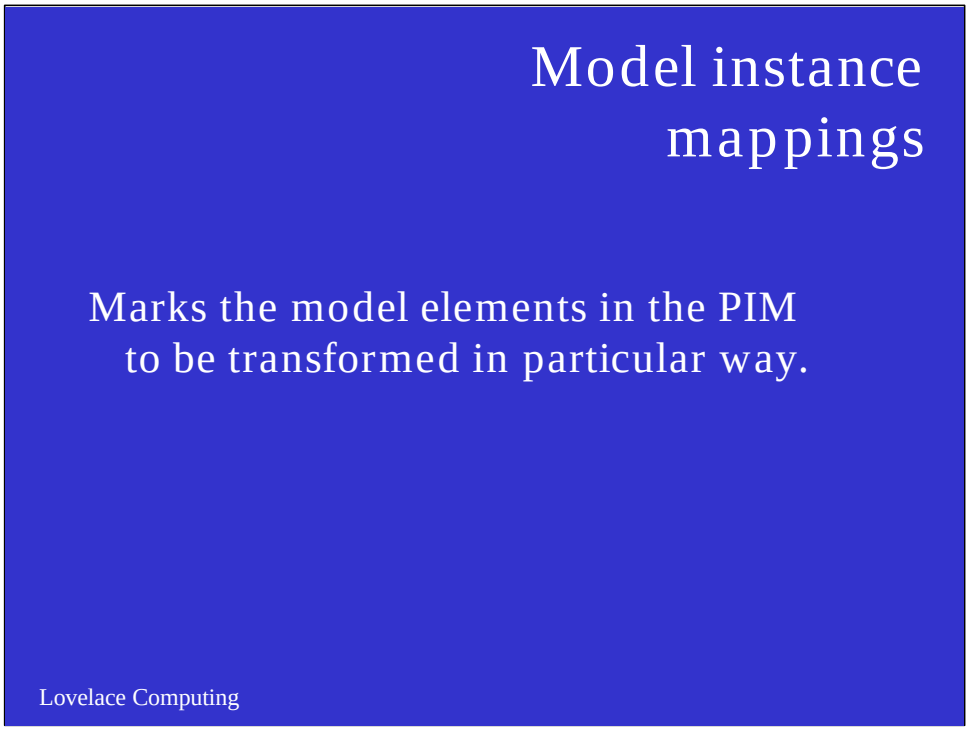
A metamodel mapping is a specific example of a model type mapping, where the types of model elements in the PIM and the PSM are both specified as MOF metamodels. In this case the mapping gives rules and/or algorithms expressed in terms of all instances of types in the metamodel specifying the PIM language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

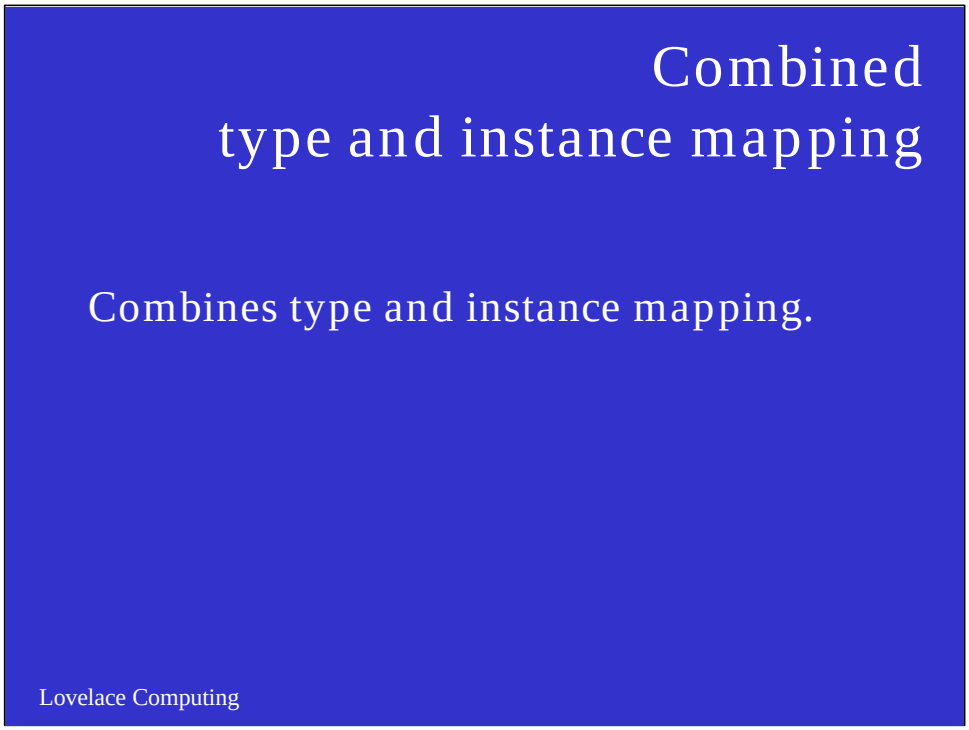


The types available to model the PSM (or even the PIM) may not be specified as a MOF metamodel. For example, the CORBA IDL language provides for the expression of types available in CORBA PSMs. In this case mappings can be expressed as transformations of instances of types in the PIM, into instances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

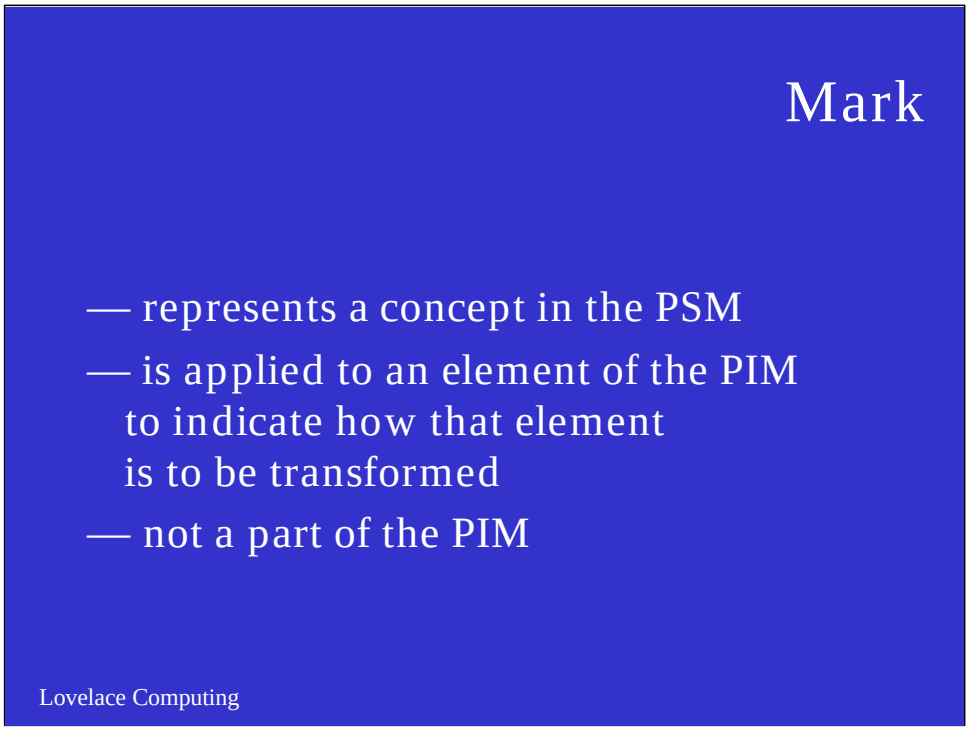
Mark

Transformation

Record

PSM

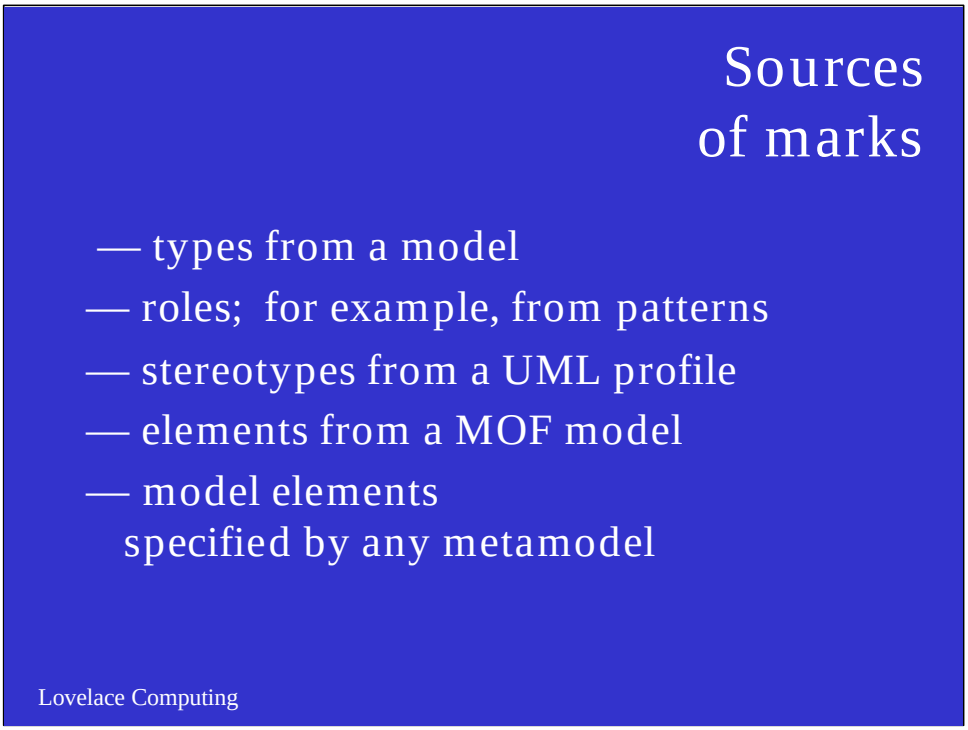
Lovelace Computing



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

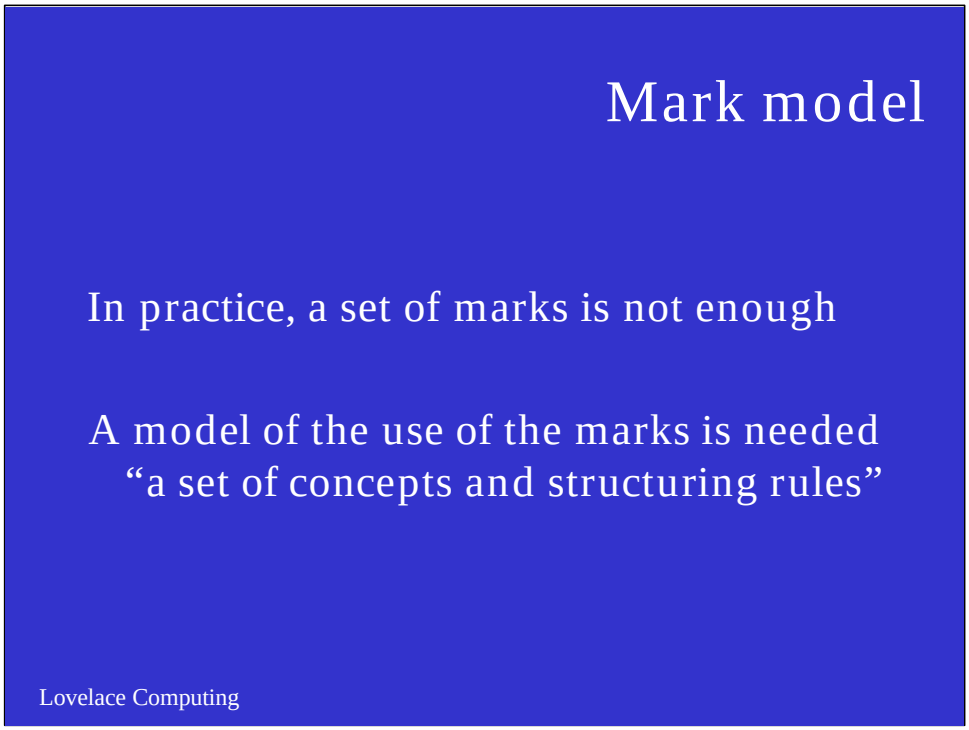
— stereotypes from a UML profile

— 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.

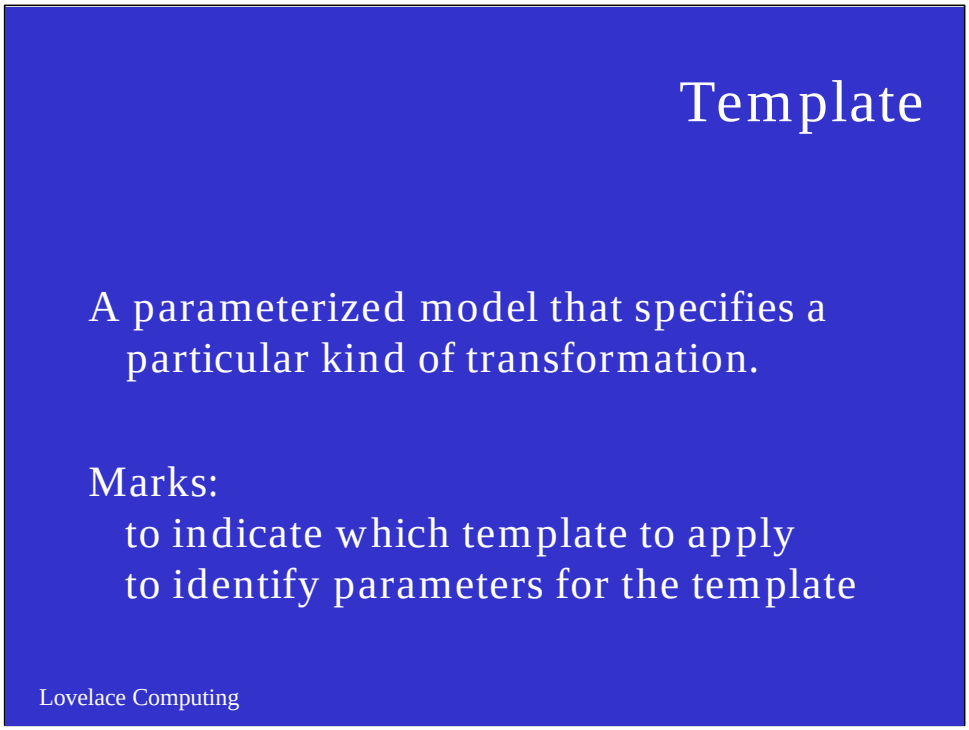
Marks may also specify quality of service requirements on the implementation. That is, instead of indicating the target of a transformation, a mark may instead simply provide a requirement on the target. The transformation will then choose a target appropriate to that requirement.



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.

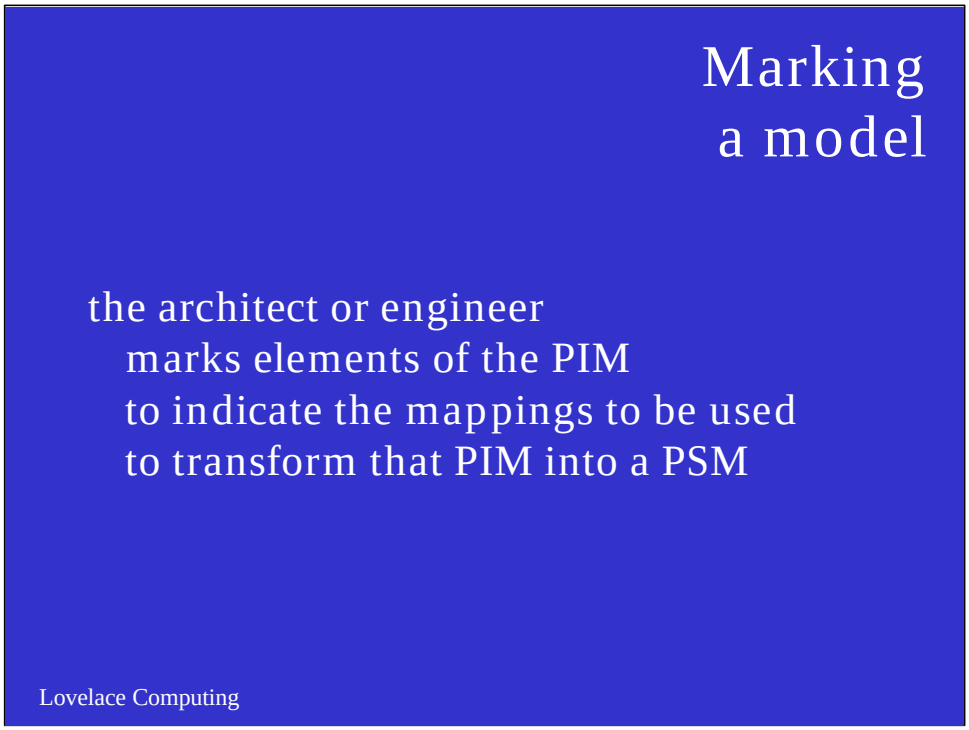
Example: A CORBA mapping might provide that a client object be prepared for a range of CORBA non-standard system exceptions or standard user exceptions and include the necessary exception handling in each case.

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.

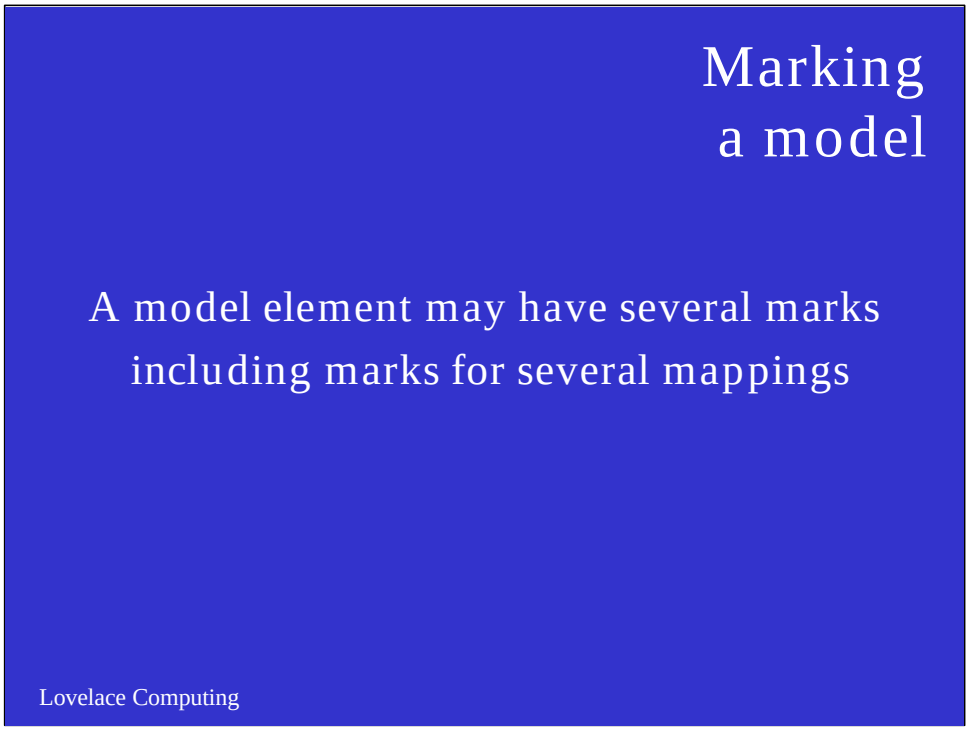
A desirable quality of a mapping language is portability. This enables use of a mapping with different tools.



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.

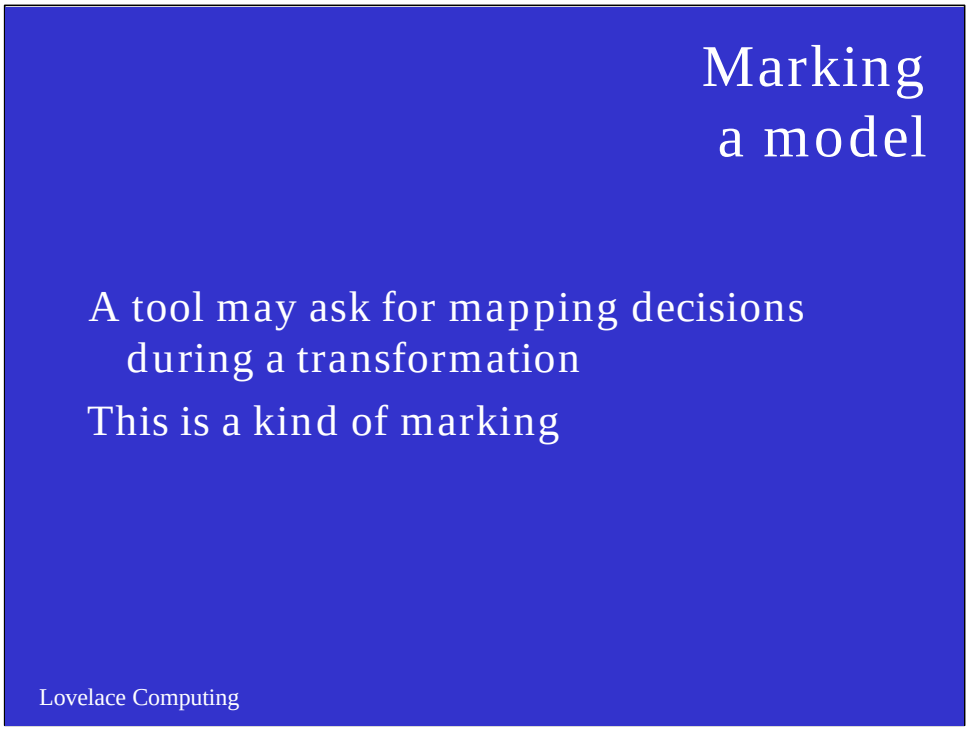
In one simple case, a PIM element is marked once, indicating that a certain mapping is to be used to transform that element into one or more elements in the PSM.

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.

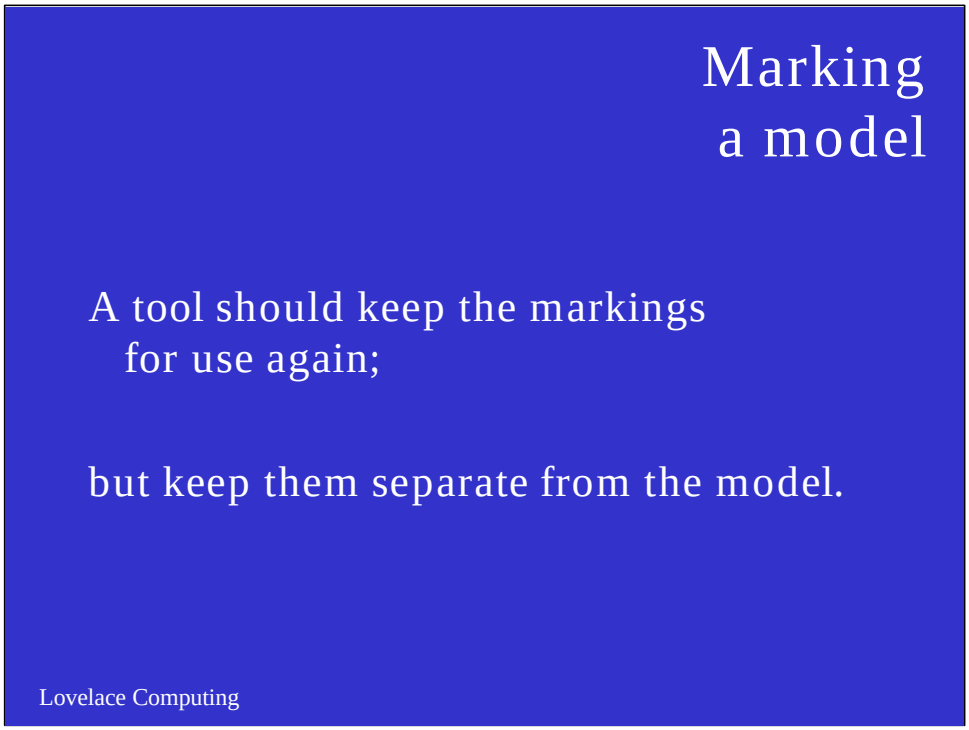


An element of the PIM may be marked several times, with *mark*s from different mappings; this indicates that the element plays a role in more than one mapping. When an element is marked in this way, it will be transformed according to each of the mappings; the result may be additional features of the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

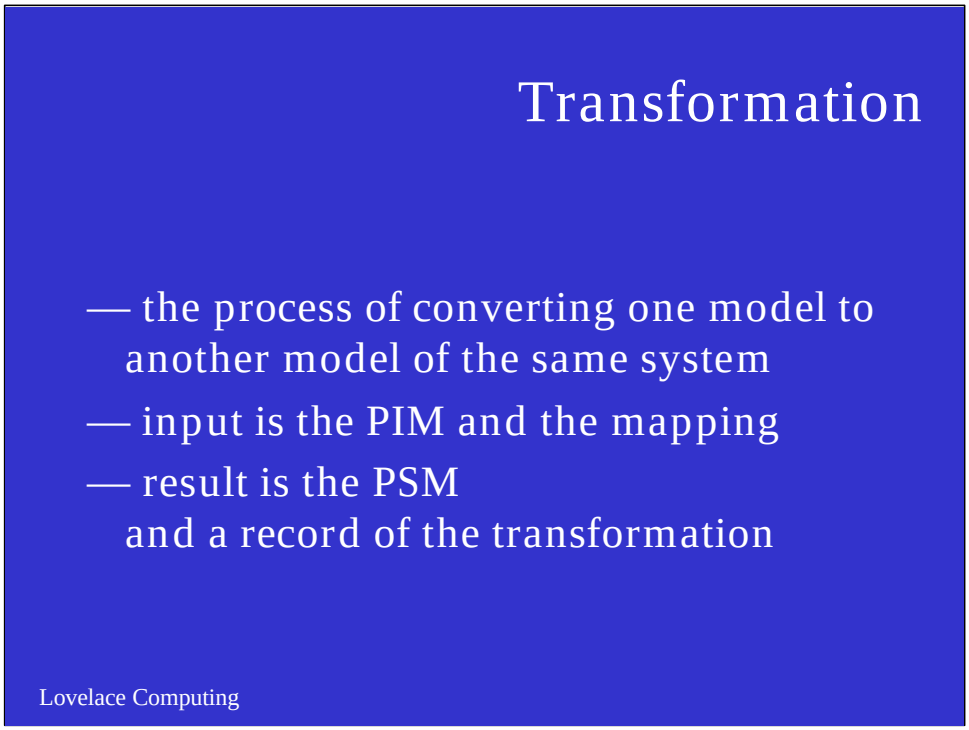
Mark

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.

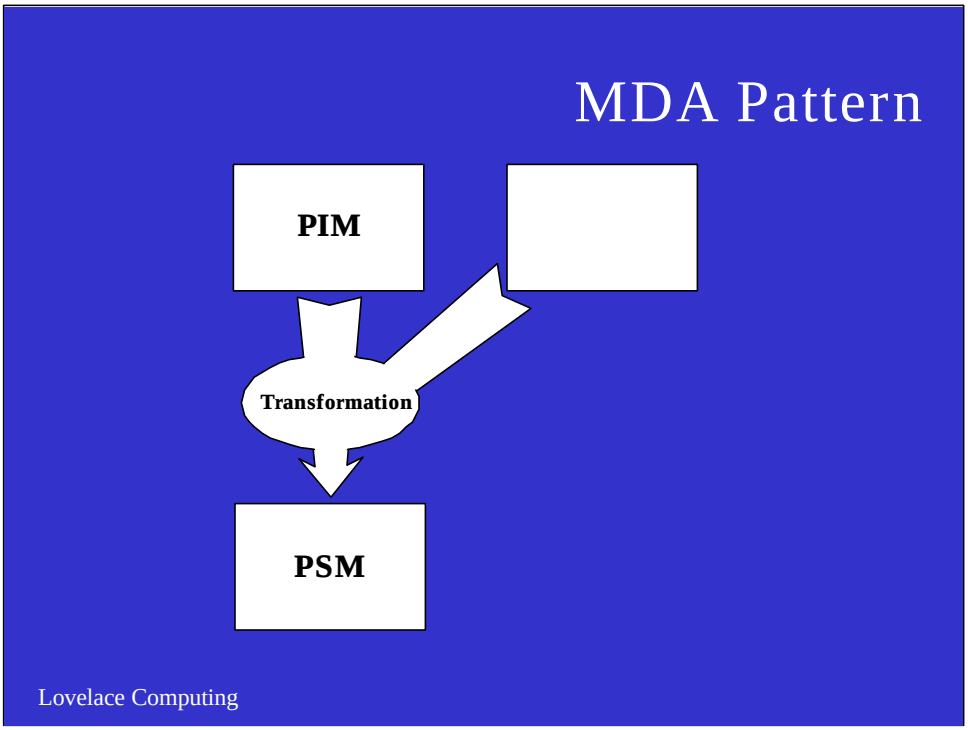
Direct to code

— 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.

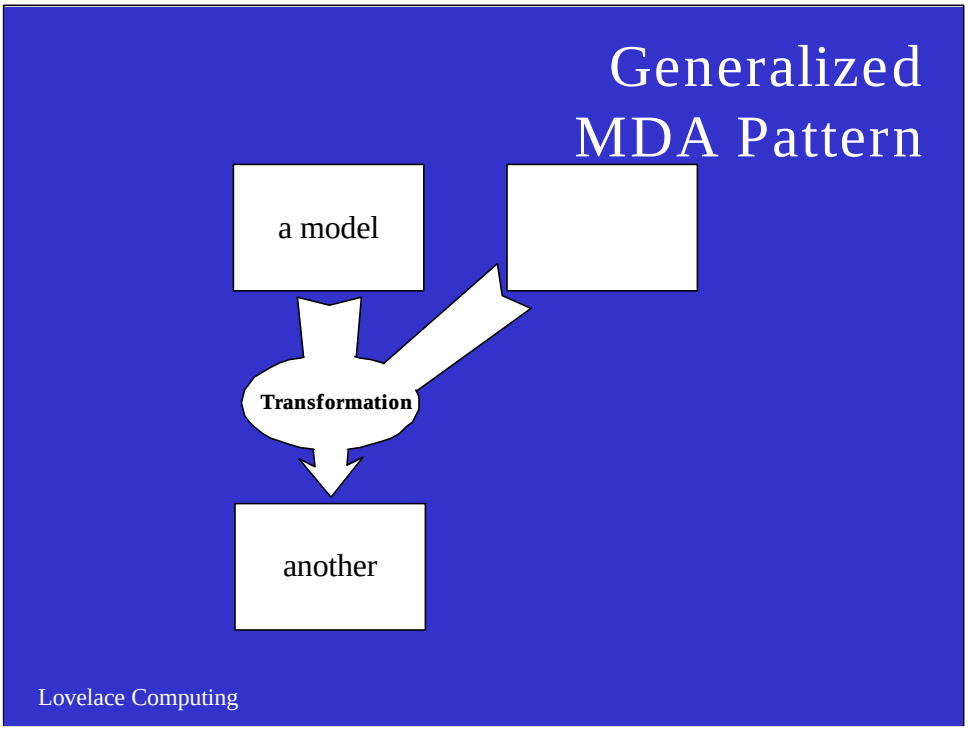


The drawing illustrates the MDA pattern, by which a PIM is transformed to a PSM.

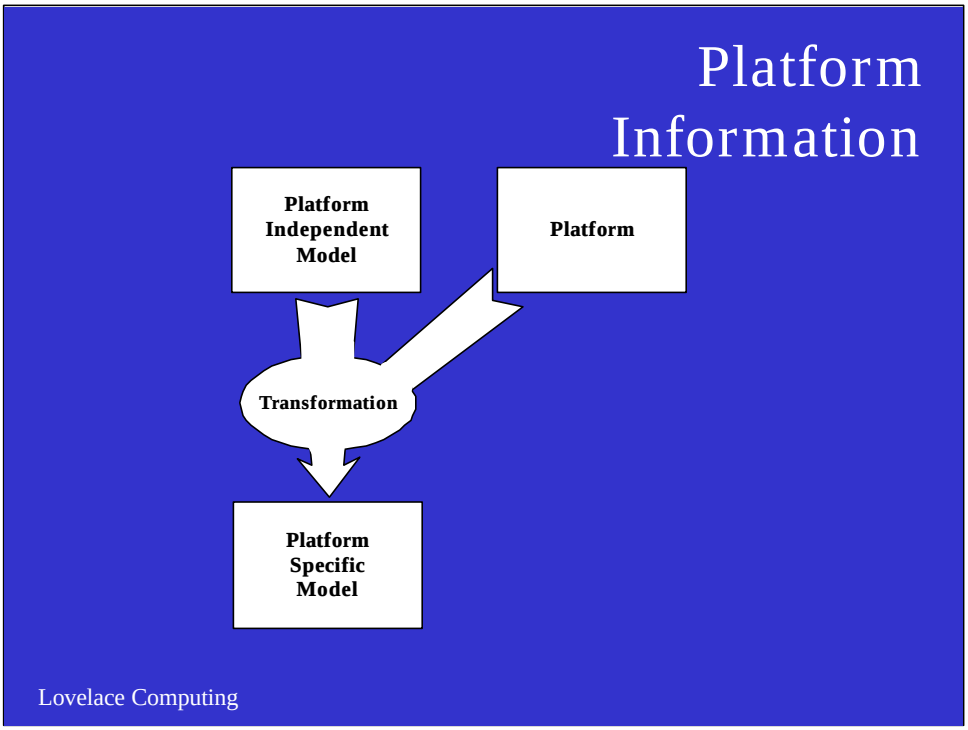
The drawing is intended to be suggestive. The platform independent model and other information are combined by the transformation to produce a platform specific model.

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.

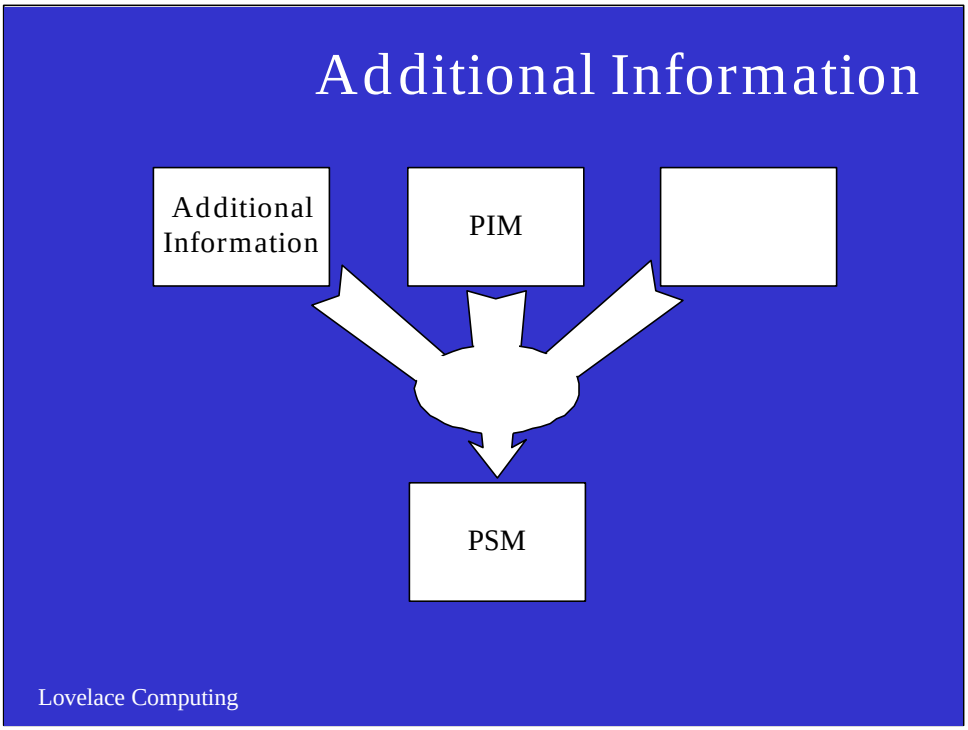
The box with no name represents what goes into the transformation, in addition to the platform independent model. This varies with different styles of MDA.



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
- Pattern application
- Model merging

Transformation approaches

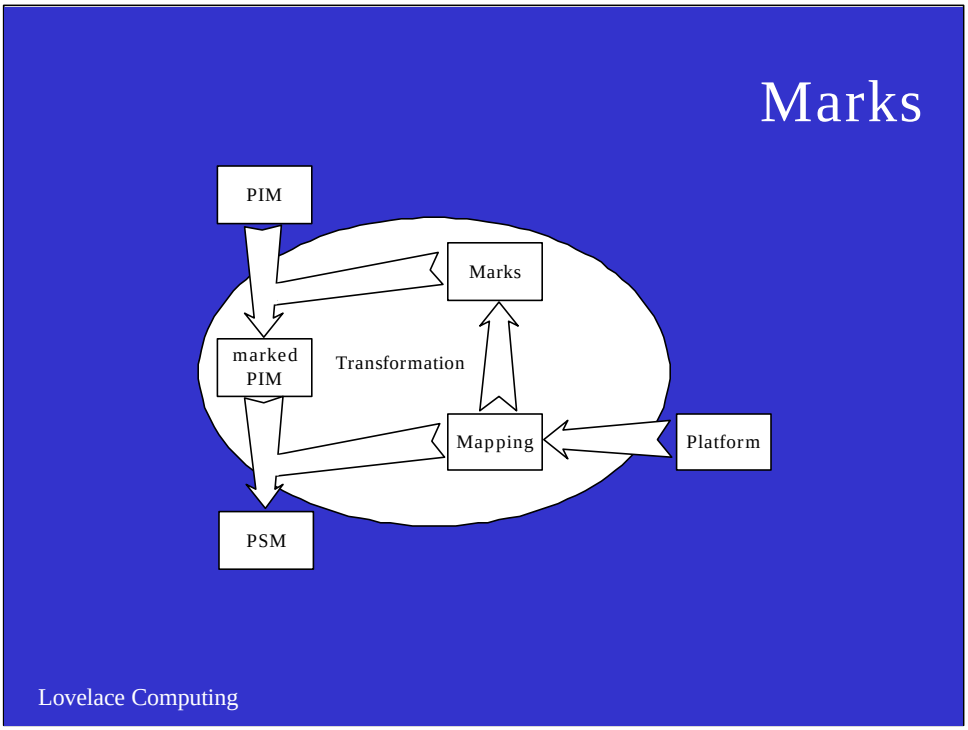
Marking

Metamodel transformation

Model transformation

Pattern application

Model merging



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

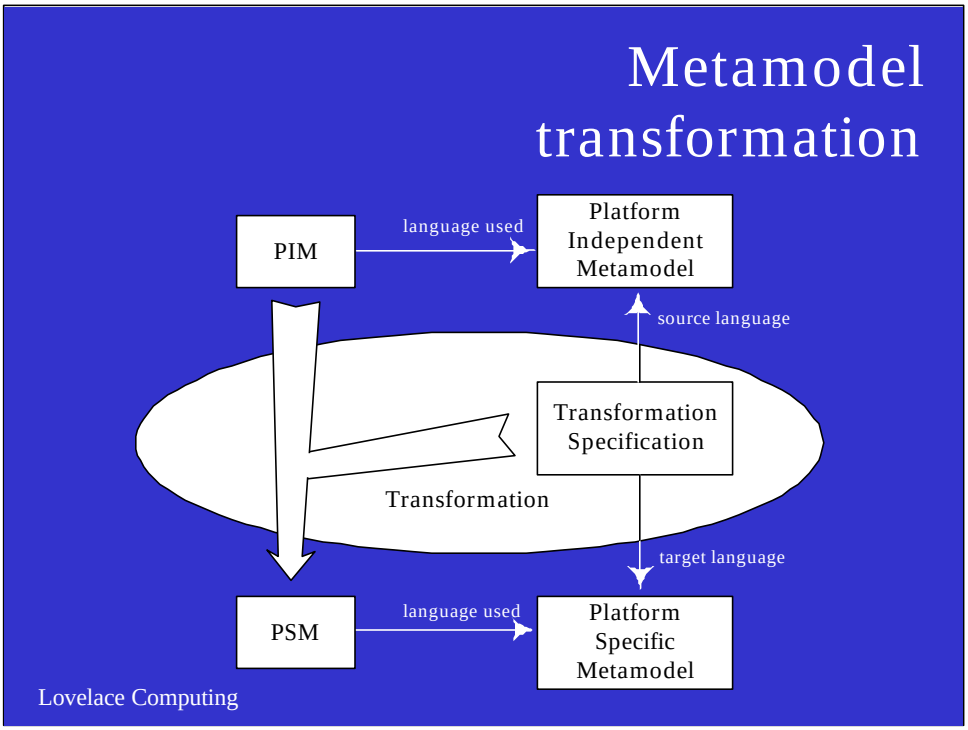
Marking

Metamodel transformation

Model transformation

Pattern application

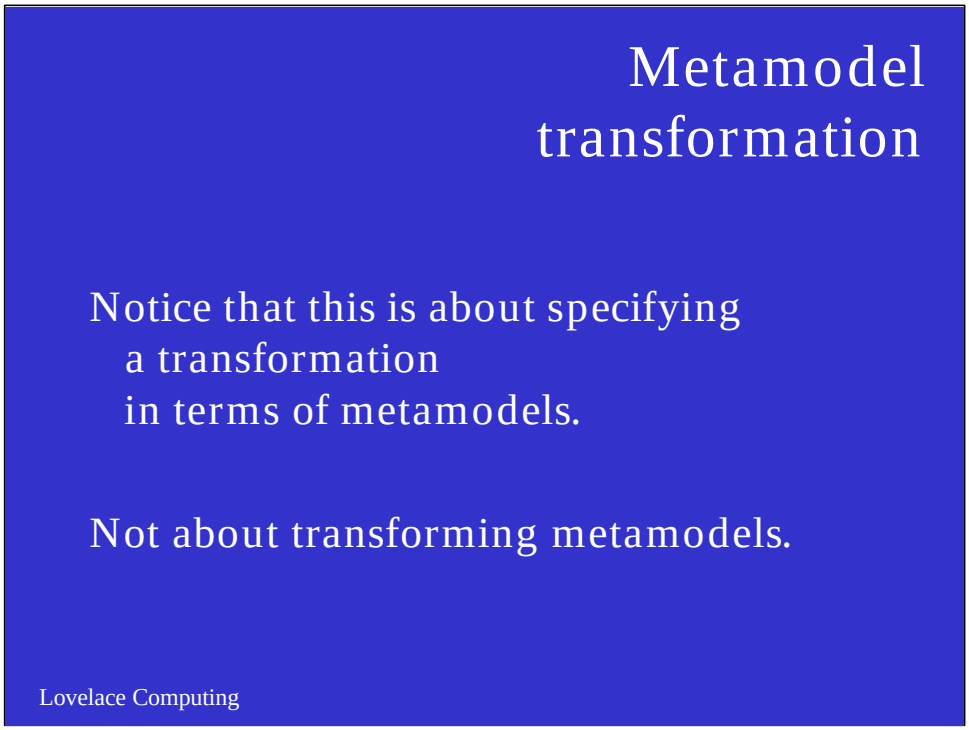
Model merging



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.

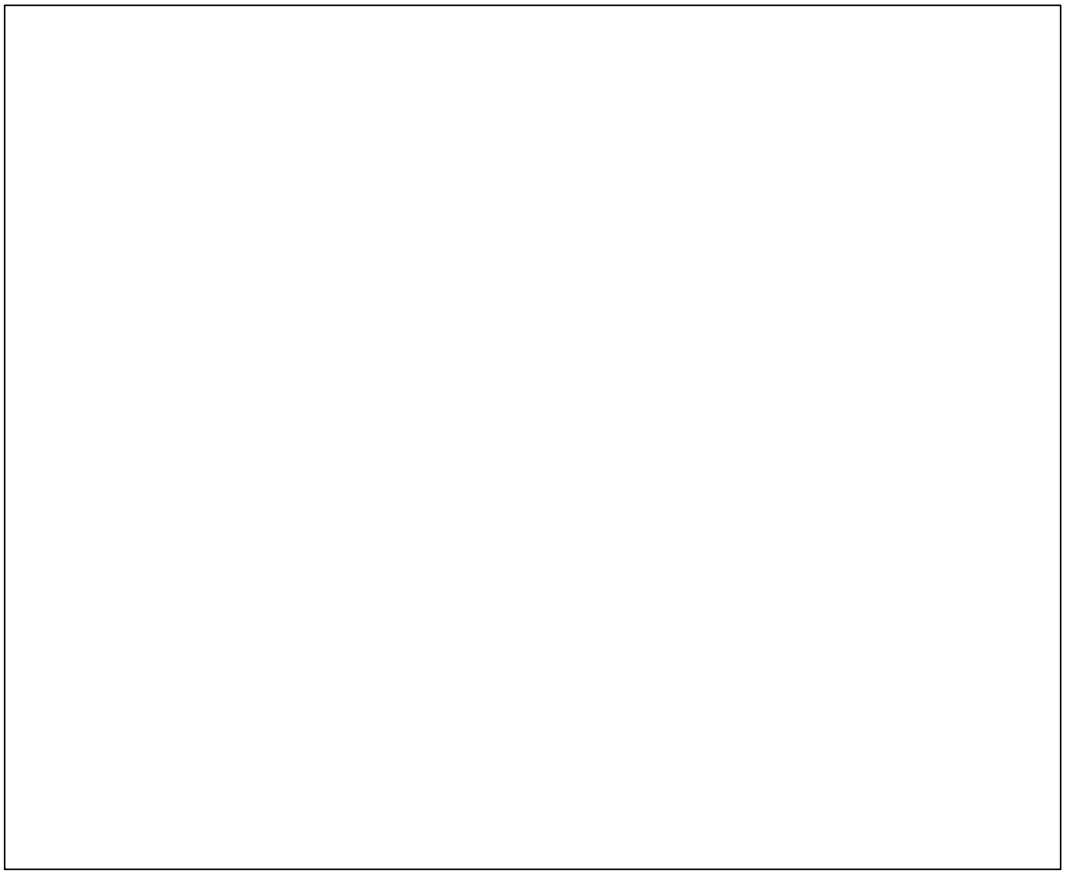


Notice that Metamodel transformation is about specifying a transformation in terms of metamodels.

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.

Metamodel transformation

Lovelace Computing



Transformation approaches

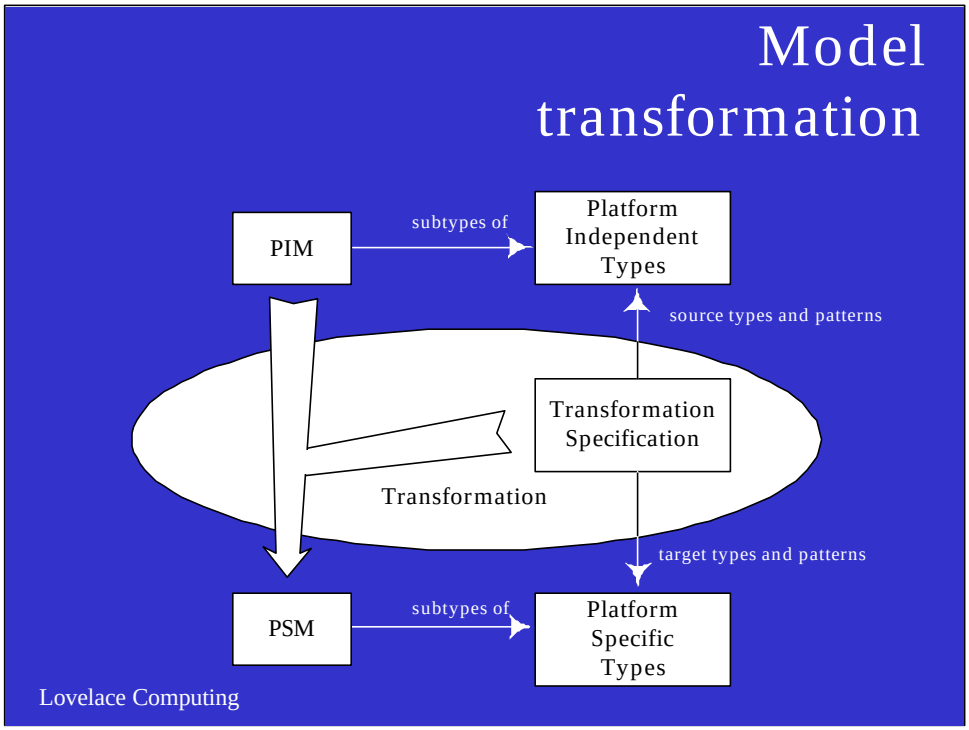
Marking

Metamodel transformation

Model transformation

Pattern application

Model merging

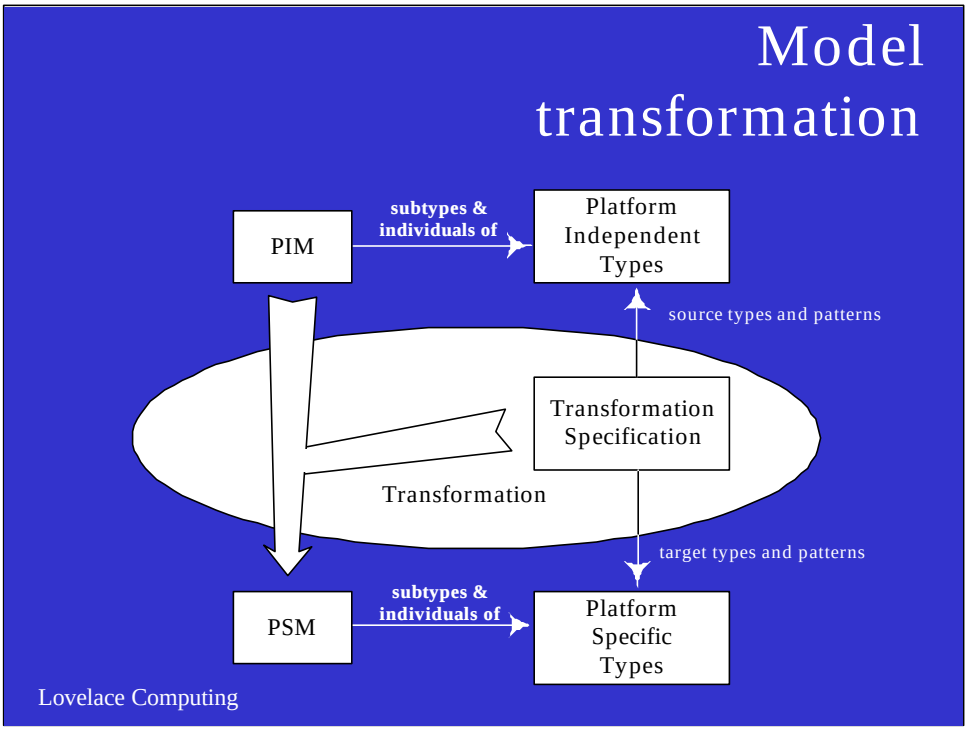


The drawing shows yet another of the ways that a transformation may be done.

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

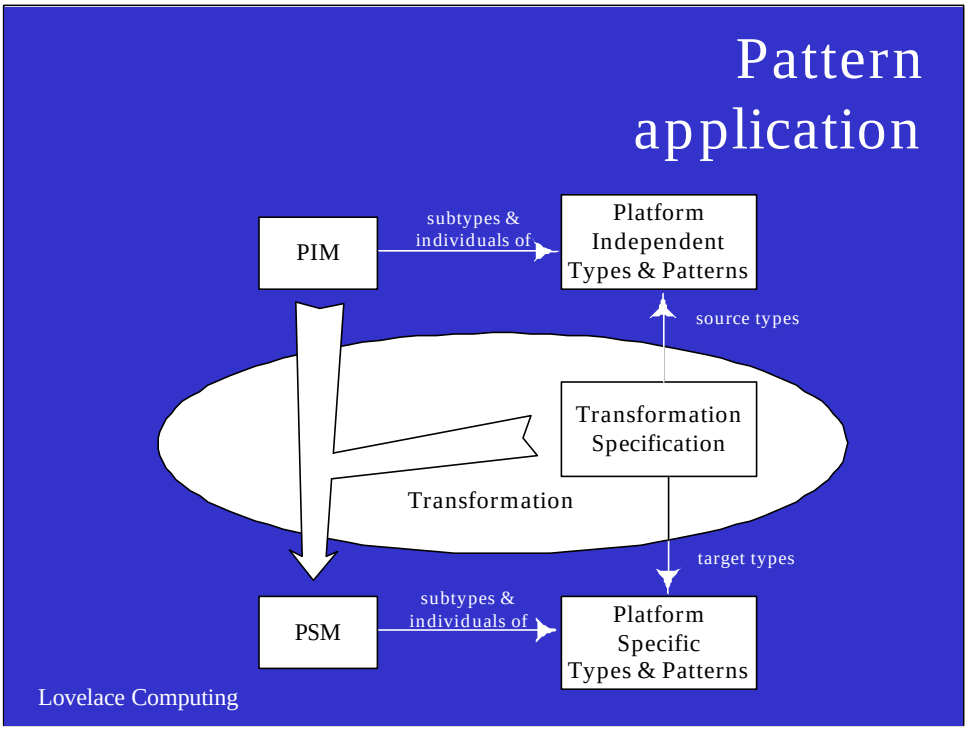
Marking

Metamodel transformation

Model transformation

Pattern application

Model merging

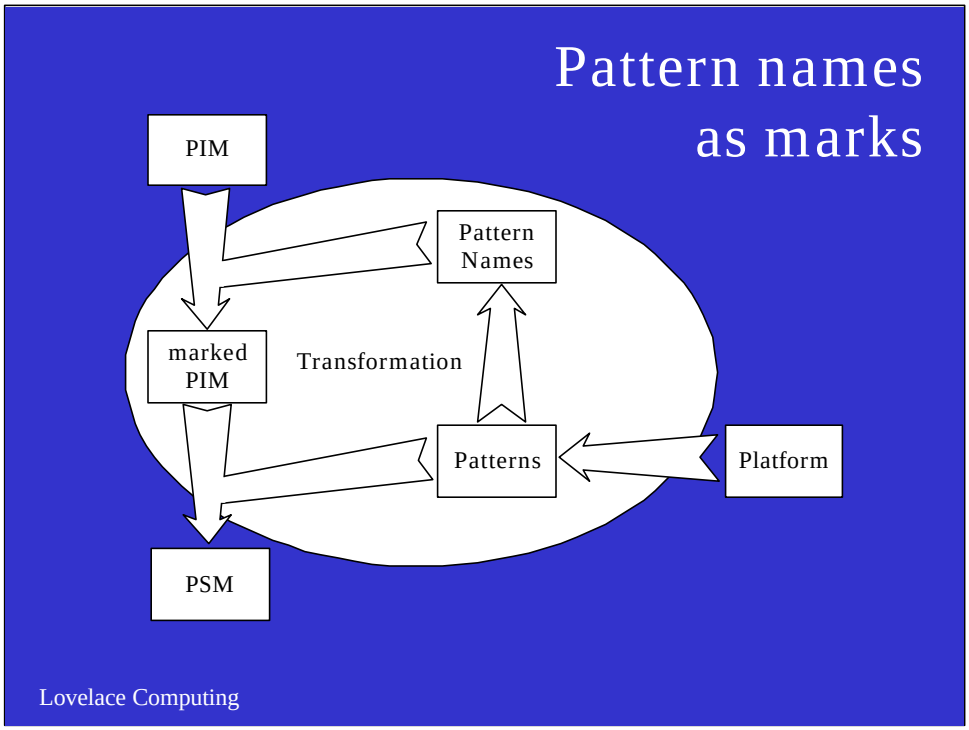


Extension of the model and metamodel mapping approaches include patterns along with the types or the modeling language concepts.

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…

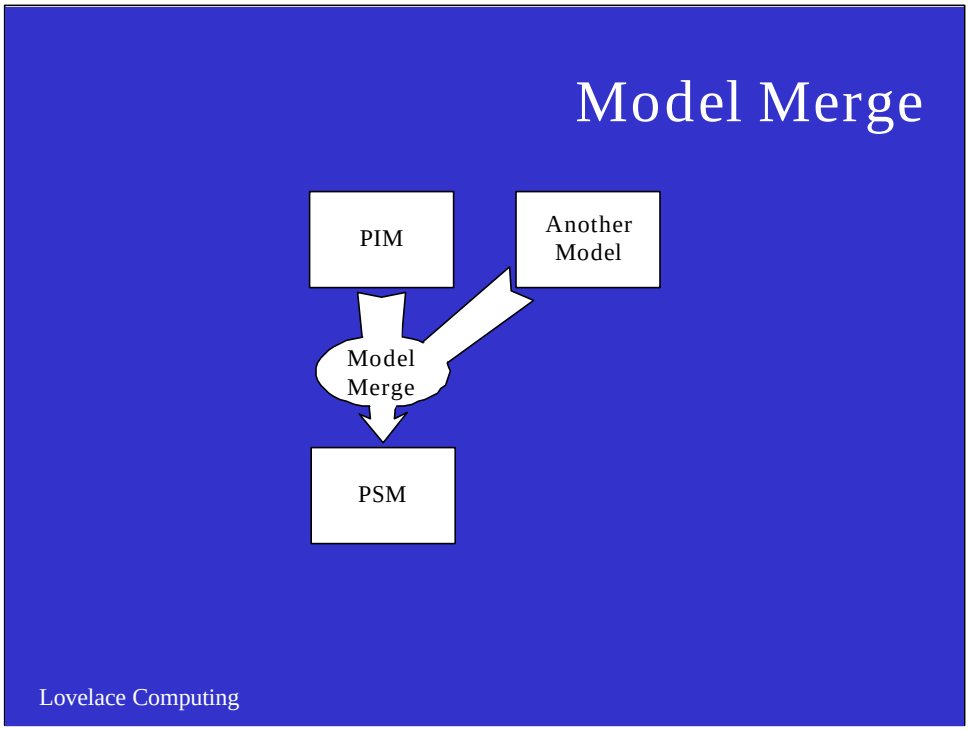
The metamodel mapping approach can use patterns in the same way.



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
- 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.

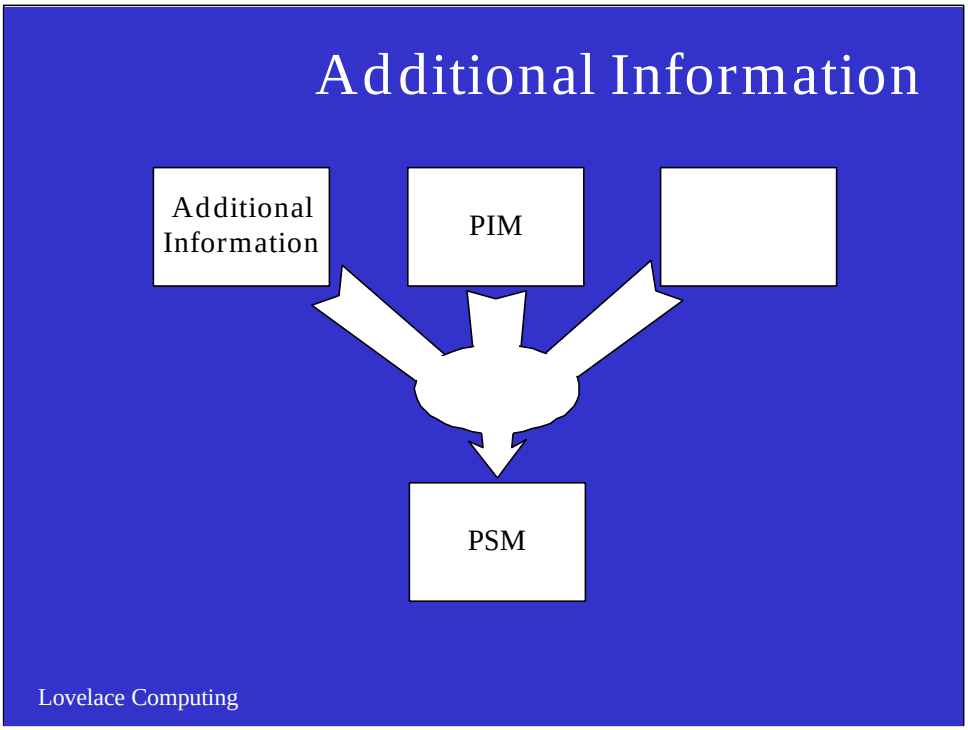
Again, the drawing is intended to be suggestive. It is also generic. There are several MDA approaches that are based on merging models.

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
- Pattern application
- Model merging

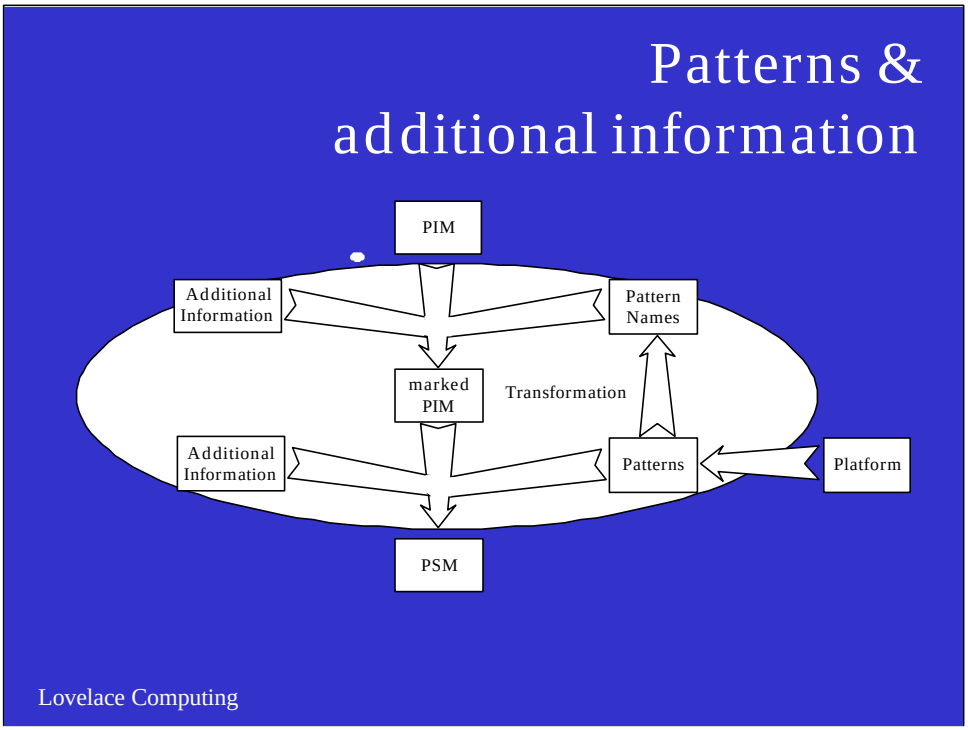


The drawing extends the simple MDA pattern to show the use of additional information.

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

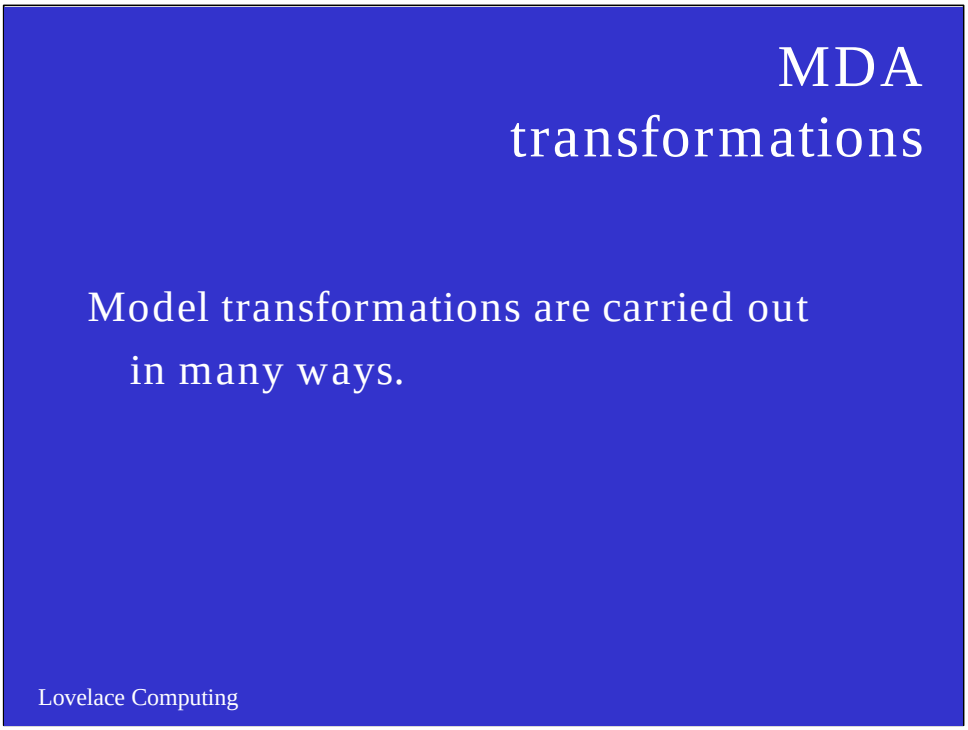
Examples: A particular architectural style may be specified. Information may be added to connectors to specify quality of service. Selections of particular implementations may be made, where more than one is provided by the transformation. Data access patterns may be specified.

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.

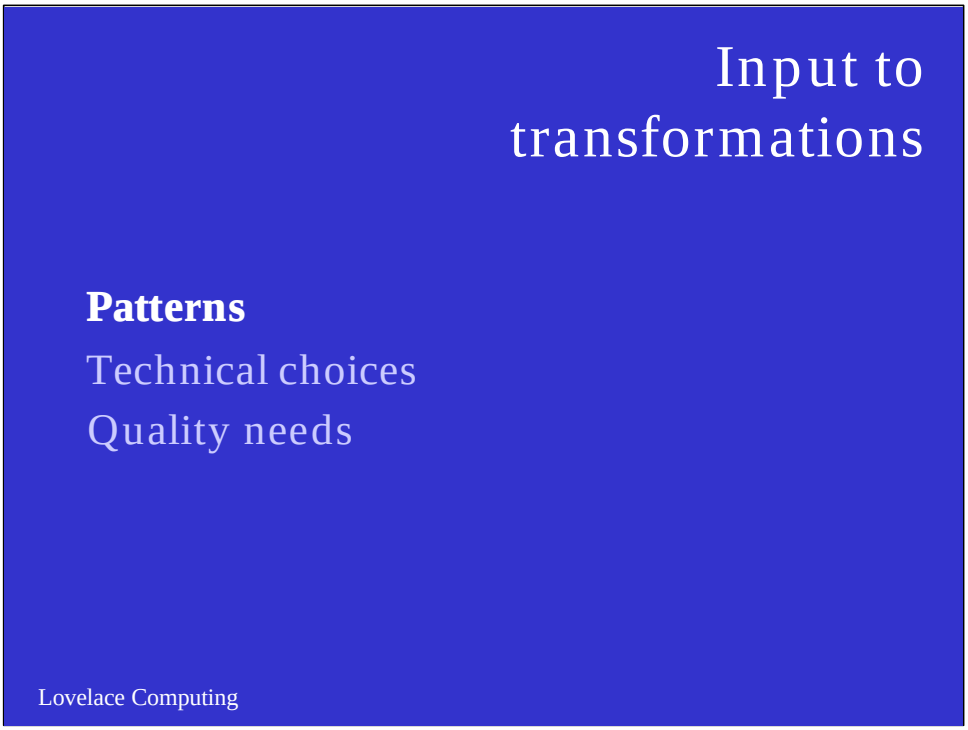
As mentioned, there may be an advantage to providing the developer a model of the generated code.

MDA transformations

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

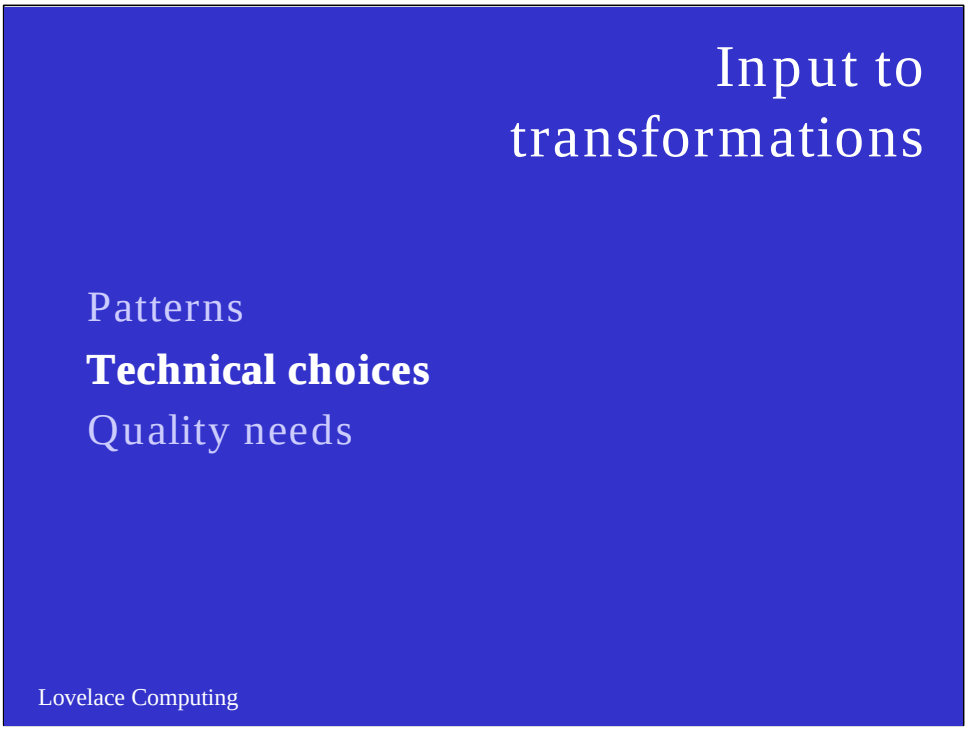
Input to transformations

- Patterns
- Technical choices
- Quality needs

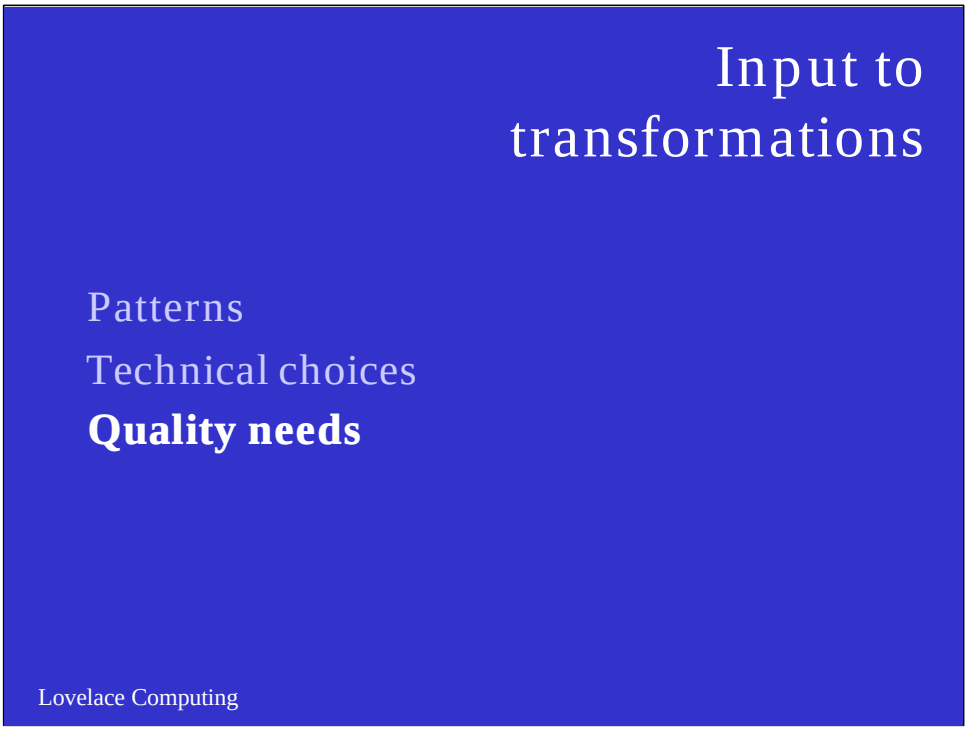


Generic transformation techniques can work with patterns supplied by the architect or builder. Different patterns, chosen by the architect, or by a transformation tool using supplied selection criteria.

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.



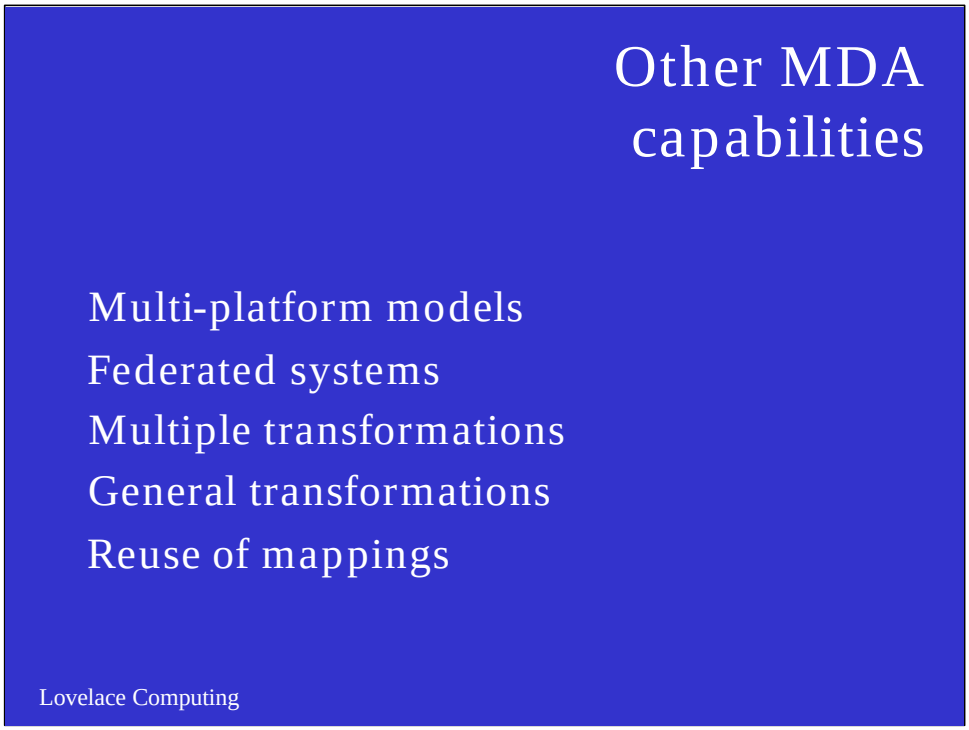
Technical choices of all kinds can be made by the architect or builder and used to guide the transformation. Technical choices might also be made by analysis tools working with the PIM, and then used in manual or automatic transformation. Most approaches will use some combination of some automated transformation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



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

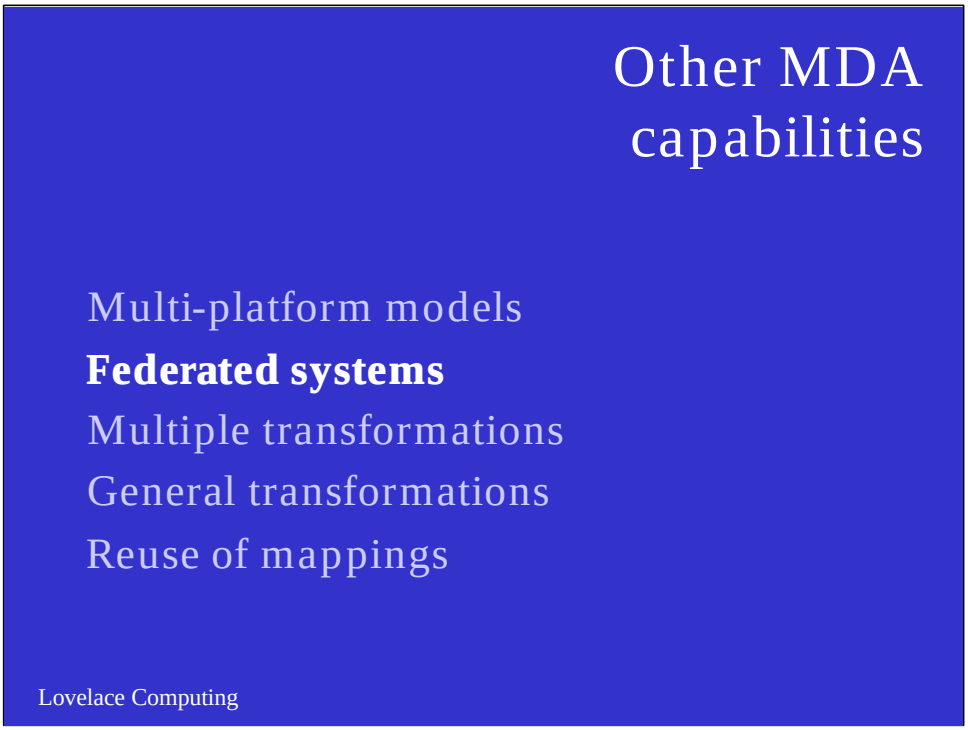
Reuse of mappings

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.

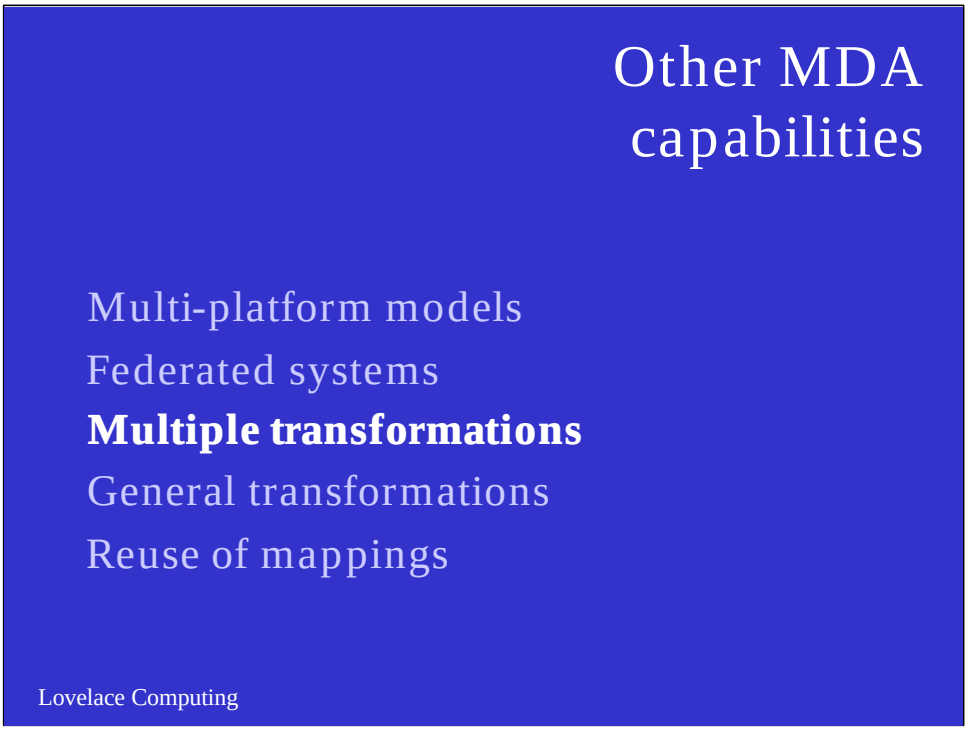
Example: A system needs to communicate with several existing systems. Several means of communication are available, IIOP, RMI, and SOAP. The architect chooses the means most suitable for each connector and marks that connector with a mark from the set for that means.



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.

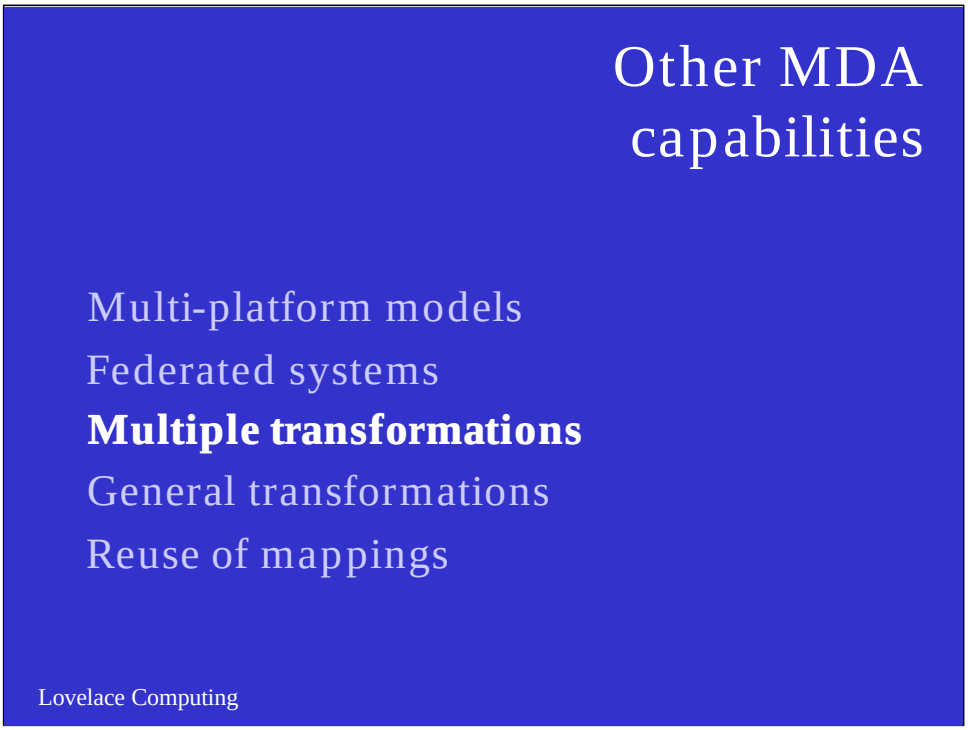
Example: Several trading partners want to share a common software design and produce interoperable implementations, each partner using a different platform.

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.



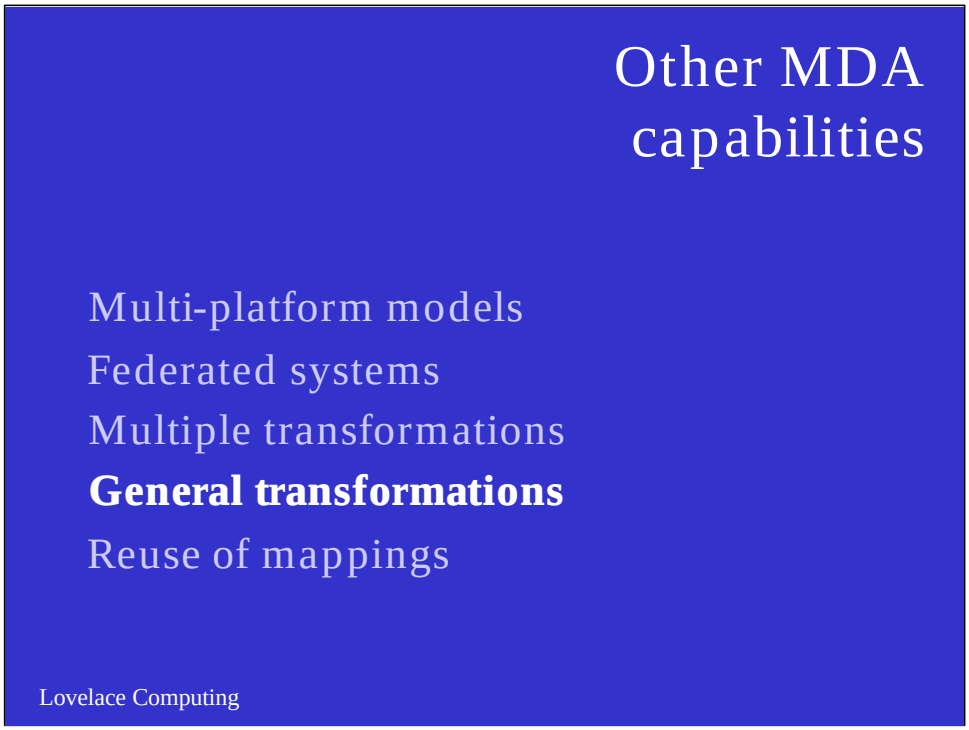
The MDA pattern includes a PIM, a platform, and a PSM. The PSM is specific to that platform. The PIM is platform independent because it is not dependent on any particular platform of that class. What counts as a PIM depends on the class of platform that the MDA user has in mind.

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.

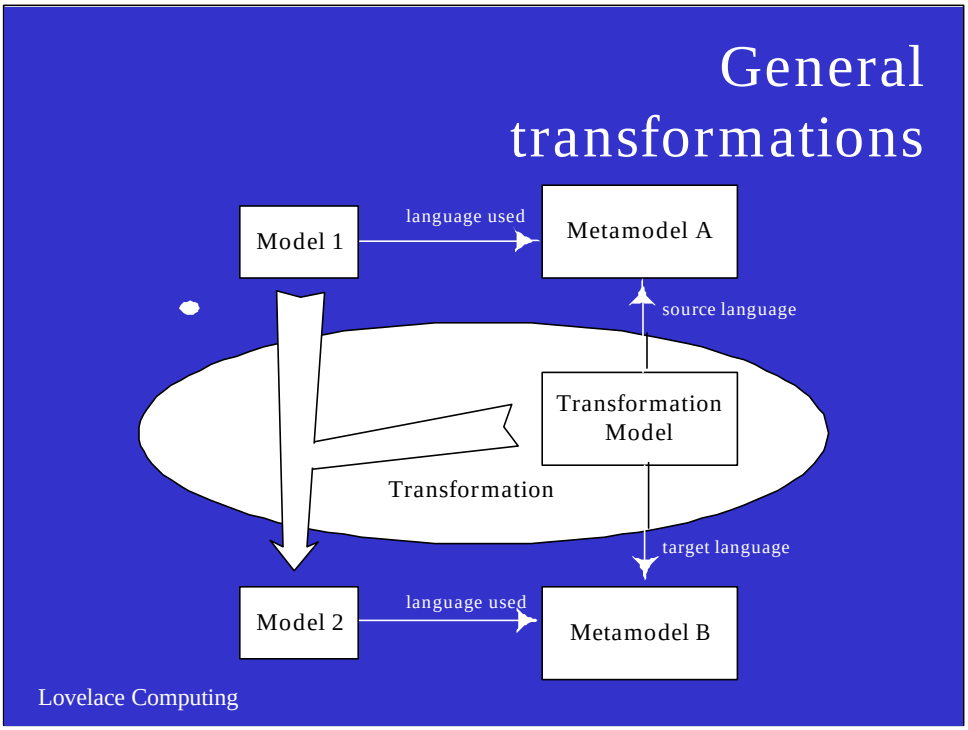


The MDA pattern can be applied several times in succession. What is a PSM resulting from one application of the pattern, will be a PIM in the next application.

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.



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.

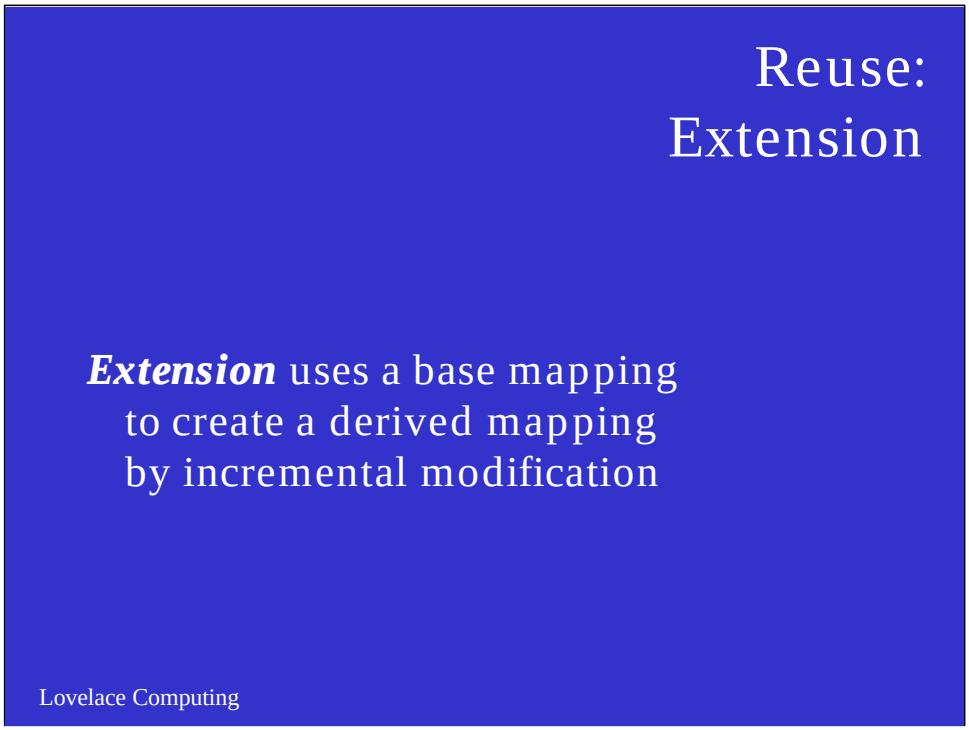
The drawing and example use metamodel mapping to illustrate the point. Any of the MDA approaches discussed in this Guide can be used for general model-to-model transformations.

Other MDA capabilities

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

Lovelace Computing

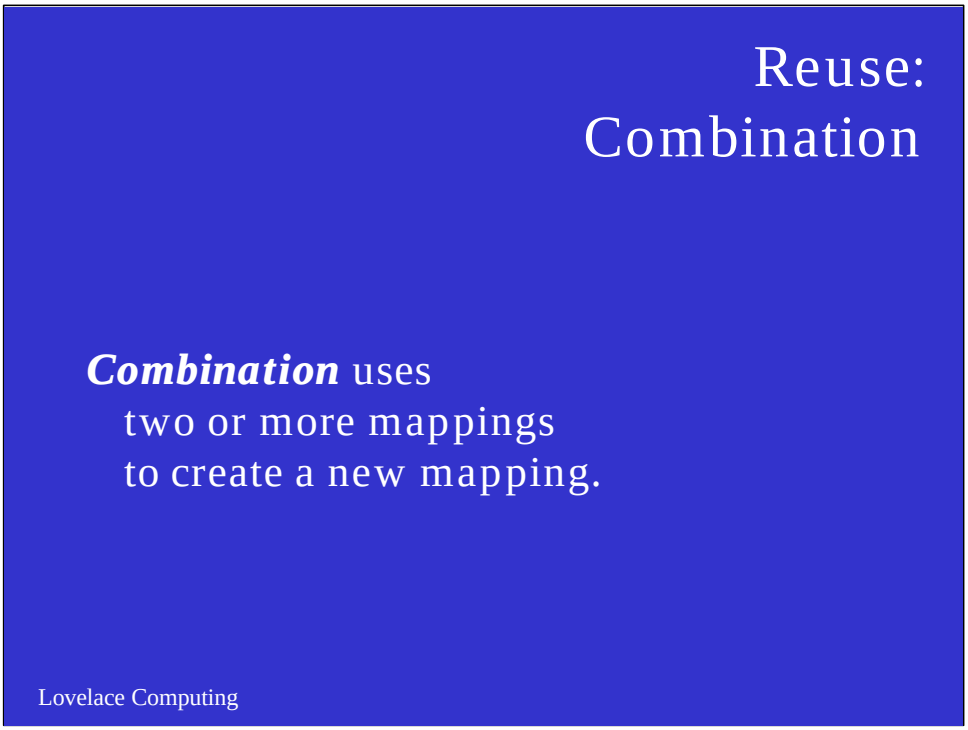
Mappings may be reused in several ways. These include extension, combination, and bridging.



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.



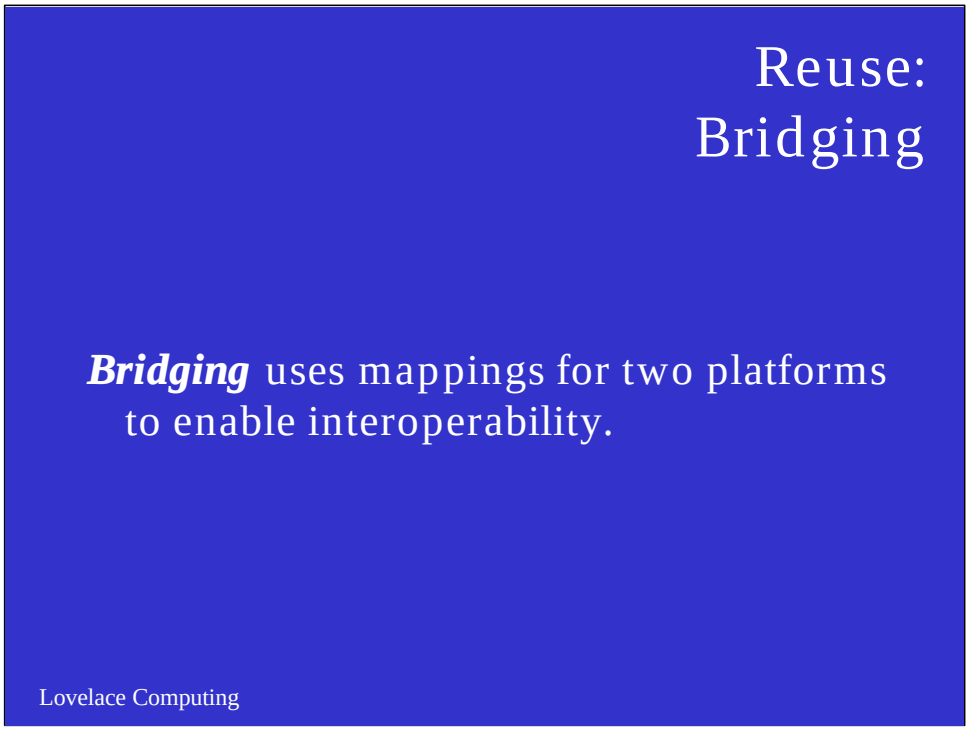
Combination uses two or more mappings to create a new mapping. The characteristics of the new mapping are determined by the mappings being combined and by the way they are combined. The effect of the application of a combined mapping is the corresponding combination of the effects of the original mappings.

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

Other MDA capabilities

Document generation

Data transformation

Test generation

...

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/

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
2908 Morgan Ave
Oakland CA 94602-3471
USA

+1 (510) 336-2545
Fax: 336-2546

joaquin@joaquin.net

www.joaquin.net