

The Fundamental Interconnectedness of All Things [1]

Kerry Raymond, CiTR/DSTC

1. Motivation

Where are enterprise distributed systems heading in the future? Technologies may come and go, but the trends are evident. Applications and systems are getting bigger and bigger. No matter what technologies are employed, we can expect to see programs with billions of lines of source, ported to hundreds of platforms, and interoperating across thousands perhaps millions of systems. Whether we use object technology, component technology, or whatever the future brings, we can be confident that systems will be compositions of many things. These things will be interconnected, interworking, and interdependent.

How will we build these huge systems? How will we find and correct their bugs? All the research points the same way: the need for rapid development and rapid evolution, the need for interoperability, the need for traceability. We need to build, evolve and maintain a universe of things, fundamentally interconnected through both their definitions and their run-time interactions.

This extended abstract outlines the Pegamento Architecture to tackle the holistic development and evolution of massive enterprise systems in Section 2. A key element, the Generator Generator, is introduced in Section 3, based on a transformation model described in Section 4. Section 5 concludes with an overview of the current implementation and future plans for the Generator prototype.

2. The Pegamento Architecture

The Pegamento Architecture is based on belief that the only scalable approach to massive enterprise systems is to substantially automate their development. However, automating code generation directly from user requirements is not feasible. Instead the Pegamento Architecture is based on the successive refinements of specifications, starting with very high-level technology-independent specifications, which are progressively refined (using generators) into more detailed specifications and into more technology-specific specifications. At each step, there may be scope for the programmer to introduce additional detail appropriate to that level of abstraction, or to request customisations in the generation of the target specification. The resultant specifications (programs or program fragments) can then be woven together to produce the code of the resultant system.

There can be many steps in the overall system refinements, but the Pegamento Architecture characterises models as being either enterprise-level (not focussing on any technology concerns), technology-unifying models (which address a class of technologies, e.g. “messaging”) and technology-specific models (e.g. Java Messaging System). Refinements will typically pass through all these kinds of models. Often there will be many possible target models for a given source model, and an enterprise distributed system will often involve elements from competing technologies, and hence from competing models. Through traceability, it is possible to understand how low-level implementations should be able to interwork, by analysing the higher-level semantic relationships that exist between them. Similar, through traceability, bugs can be traced to their original source, and corrected through re-generation.

Although generators are the basis for the Pegamento Architecture, our initial experiences with prototyping have shown us that writing them by hand is both slow and error-prone. Using technologies such as MOF [2] and UML [3], it is easy to construct models, and so the development of generators is likely to become the bottleneck in systems development. Therefore, the Pegamento Architecture incorporates a Generator Generator to enable the generators themselves to be generated based on declarative descriptions of refinements/transformations between models.

Using the Pegamento Architecture, the programmer first prepares enterprise-level specifications in the enterprise models of choice. The programmer then selects the desired implementation technologies, which must be reachable from the enterprise models through some sequence of model refinement/transformation. The Generator Generator is then used to create a series of generators to perform each of refinements/transformations (if they don't already exist). The individual generators are then executed in sequence; the programmer may choose to add additional information or customisation at each intermediate model. The final system can be built and run using the chosen technologies.

Recently OMG announced its new Model-Driven Architecture [4] which is extremely similar to the Pegamento Architecture.

3. The Generator Generator

The purpose of the Generator Generator is to enable correct and efficient generators to be implemented quickly and (as far as possible) declaratively. Figure 1 illustrates an example of using the Generator Generator to convert between the Enterprise Distributed Object Computing model (EDOC, developed jointly by the Elemental and Pegamento projects and currently subject to an OMG adoption vote) [5] and Pegamento's workflow product Breeze [6]. To assist in interpreting Figure 1, note that

boxes that are shaded are those elements which some programmer/modeller must write, while the unshaded elements are generated.

At the top of Figure 1, there are 3 models: the EDOC model, the Breeze model, and the Transform model. All of these models are expressed using the Meta-Object Facility (MOF), an OMG modelling standard [2] developed by DSTC and Unisys. The EDOC model includes the concepts needed to express some aspects of enterprise systems, such as business processes, expressed in terms of activities and their inputs and outputs and the data flows that connect those inputs and outputs. The EDOC model is a technology-independent model. The Breeze model describes the workflows that can be directly implemented using the Breeze workflow product, expressed in terms of tasks, conditional tasks (shown as “If”) and

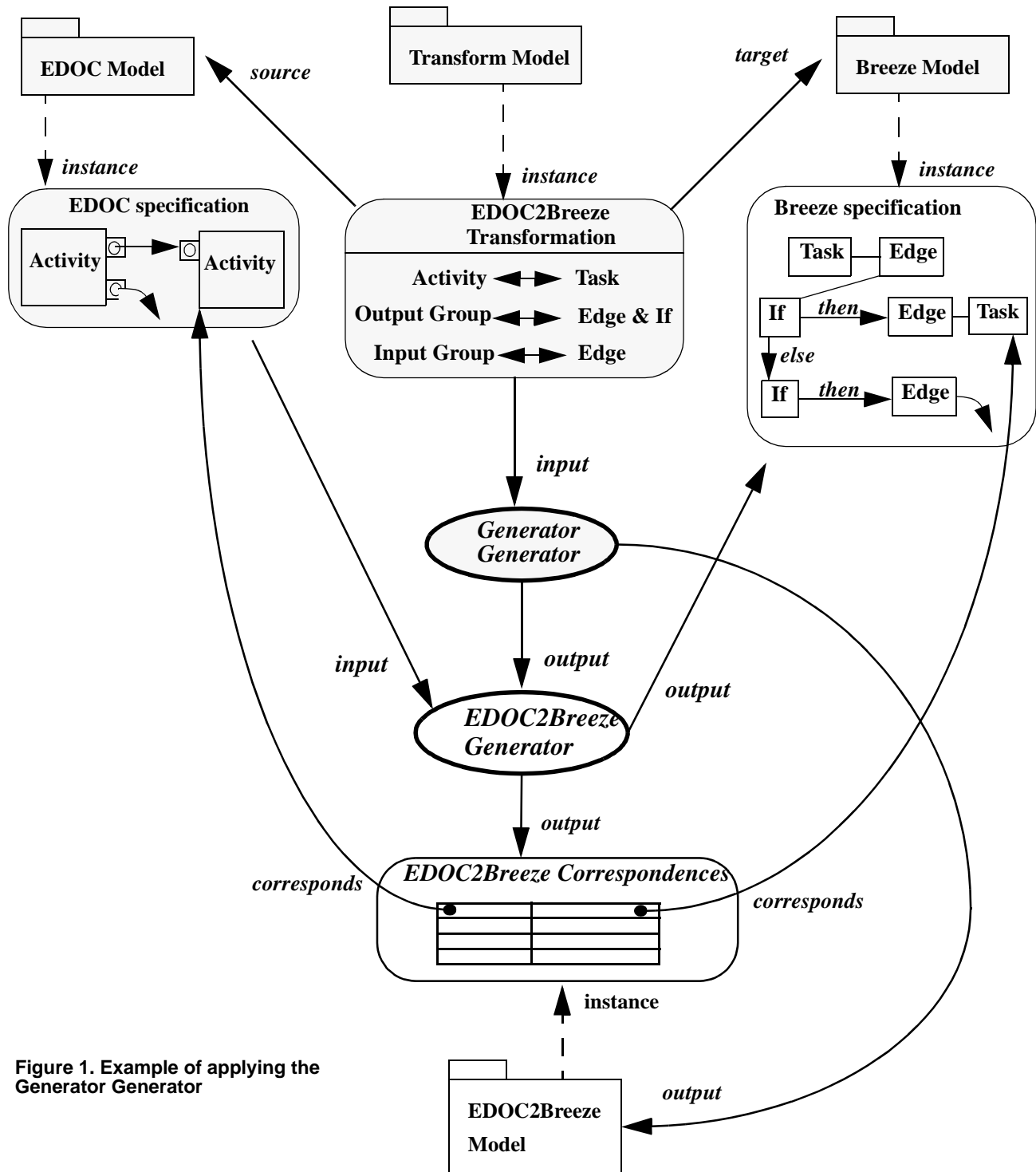


Figure 1. Example of applying the Generator Generator

edges that connect tasks. The Breeze model is technology-specific. The Transform model includes the concepts needed to describe the transformations between arbitrary MOF models, and is described more fully in Section 4.

Below these models in Figure 1 are examples of the instances of the EDOC, Breeze, and Transform models. The EDOC specification (an instance of the EDOC model) shows an activity with two possible groups of outputs, one of which initiates a second activity. The Breeze specification (an instance of the Breeze model) illustrates a Breeze workflow equivalent to the example EDOC specification, showing two tasks, two conditional tasks (shown as “If”) and three edges that control the initiations of tasks. The EDOC2Breeze Transformation (an instance of the Transform model) describes how to convert from the source EDOC model to the target Breeze model. It is important to note that the transformation is described in terms of the models, and not in terms of the instances (the EDOC and Breeze specifications). The EDOC2Breeze transformation consists of rules, each of which comprises some input elements from the source EDOC model and some output elements from the target Breeze model and the instructions on how to transform the input to the output (the instructions are too detailed to show in Figure 1). Each EDOC activity will be transformed into a Breeze task. Each EDOC output group will be transformed into a Breeze edge to a Breeze condition task, which is used to enable the appropriate subsequent task. Each EDOC input group will be transformed into a Breeze edge used to receive the control signals to initiate the EDOC activity’s task. The instructions will establish the values of the attributes and links within the Breeze specification based on the attributes and links within the EDOC specification.

Having described how to transform the EDOC model into the Breeze model as the EDOC2Breeze Transformation, the EDOC2Breeze Transformation is then used as the major input to the Generator Generator. Other inputs will include user-customisation choices (e.g. object granularity, time/space trade-off preferences, preferred optimisations). There are two outputs from the Generator Generator: the EDOC2Breeze generator and the EDOC2Breeze model (to be discussed in Section 3.1).

As shown in Figure 1, the EDOC2Breeze generator is a program which reads an EDOC specification as input and produces a corresponding Breeze specification, based on the transformations described in the EDOC2Breeze Transformation. This generator can be used to transform any EDOC specification into the corresponding Breeze specification, and by being generated itself, speeds up the systems development process and reduces the potential for bugs due to human carelessness (in the same way as high-level languages are believed to reduce bugs compared with programming directly in assembly or machine language). Of course, the generator cannot be guaranteed to be bug-free, as there may be errors in the source or target models or in the transformation rules provided, or in the Generator Generator.

3.1 Tracking correspondences

While transforming an EDOC specification into a Breeze specification meets the need for generation of models, it does not need the parallel need for traceability to support bug fixing, reverse engineering, and round-tripping. Therefore, as each transformation is performed by the EDOC2Breeze generator, the correspondences between the input EDOC elements to the generated Breeze elements are recorded as instances of the EDOC2Breeze model, generated by the Generator Generator. This correspondence information is not just an output of the EDOC2Breeze generator, but is also information used within the generation process.

A very common transformation pattern is shown in Figure 2. In this pattern, two classes and an association between them is transformed into 2 different classes with a different association between them, but the transformation is structure-preserving. This means that if A1 and B1 are associated by C in the source model, then A1’s corresponding element X1 will be associated by Z with B1’s corresponding element Y1 in the target model.

Therefore, the transformation between association C and association Z must be written terms of the correspondences established between the previous transformations between classes A and X, and between classes B and Y. This illustrates how the correspondence information is used within the generator itself, as well as being created by it.

4. The Transform Model

The Transform model was developed in the Pegamento project, but draws some ideas from the Common Warehouse Metadata model [7] which includes models for recording correspondences between instances of MOF models.

A transformation consists of a number of ordered steps. Each step is a Package2Package transformation, and identifies the source and target models. Each Package2Package transformation consists itself of a series of Classifier2Classifier transformations; classifiers being an abstract concept in the MOF of which Class and Association are the concrete types. Each Classifier2Classifier identifies its input and output Classes and Associations. All inputs may have a precondition; transformation will not occur unless all preconditions are satisfied. Each input Class must indicate if the transformation applies also to subtypes or only to direct instances of the input Class. Each output may specify a postcondition (often

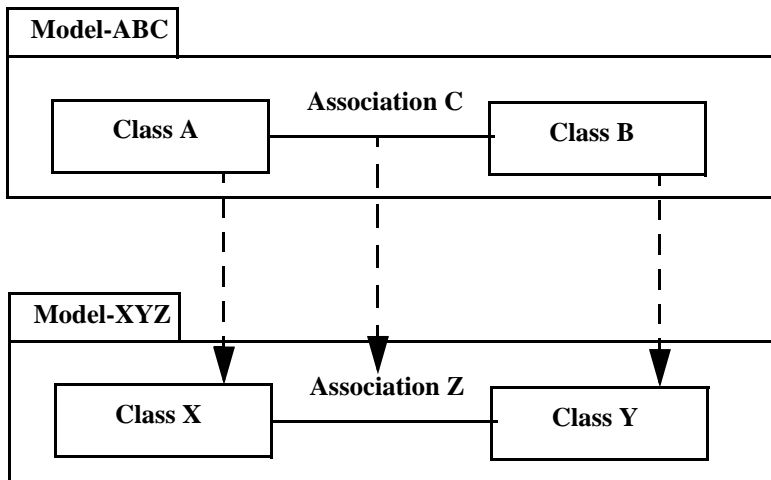


Figure 2. Common transformation pattern

reflecting constraints in the target model); the results of the transformation are discarded if the output postconditions are not satisfied. A Classifier2Classifier describes the transformation between its inputs and outputs either by user-specified functions (attributes of the Classifier2Classifier class) or by containing Feature2Feature transformations.

Feature2Feature transformations describe how its input MOF features (Attributes and AssociationEnds) are transformed into output MOF features. The features used as input/output must belong to the classes and associations identified as inputs and outputs in the enclosing Classifier2Classifier transformation. In addition, there are Feature2Classifier and Classifier2Feature transformations which enable models of significantly different granularities to be transformed.

5. Current Status, Future Work and Conclusions

The current implementation of the Generator Generator is built using DSTC's dMOF product [8], the Visibroker ORB, and Java. The functions performed by transformations can be on a number of built-in functions (e.g. copy, assign to constant) or may be a user-supplied piece of Java code. The created generator program establishes variables with "obvious" names for the use of user-supplied code. Approximately 80% of the functionality of the Generator Generator has been implemented.

Future work involves the competition of the prototype functionality. More example transformations will be developed to determine the appropriate expressive power for the Generator Generator, with corresponding extensions to the Generator Generator's built-in functions. Finally, it is hoped to describe the Generator Generator as a transformation between MOF models and a Generator model. This will enable the Generator Generator itself to be generated, and will speed its development by enabling new versions to be bootstrapped from the previous versions (similar to the technique used to build new versions of the DSTC's dMOF from previous versions).

In conclusion, the Generator Generator is a key element in constructing and evolving future enterprise distributed systems in an architecture based on generation, customisation, and traceability. The Generator Generator is at the heart of the fundamental interconnectedness of all things.

6. References

- [1] Douglas Adams, "Dirk Gently's Holistic Detective Agency", William Heinemann Ltd, 1987.
- [2] Object Management Group, "Meta-Object Facility", 2000, OMG formal/2000-04-03.
- [3] Object Management Group, "Unified Modeling Language (UML)", 2001, OMG formal/2001-09-67.
- [4] Object Management Group, "Model Driven Architecture: The Architecture Of Choice for a Changing World", 2001, www.omg.org/mda/
- [5] EDOC Partners, "Enterprise Distributed Object Computing", 2001, OMG ad/01-06-09 and ad/01-08-18.
- [6] DSTC, "Breeze: workflow with ease", www.dstc.edu.au/Research/Projects/Pegamento/Breeze/breeze.html
- [7] CWM Partners, "Common Warehouse Metamodel", OMG ad/2001-02-{01,02,03}
- [8] DSTC, "dMOF: an OMG Meta-Object Facility Implementation", www.dstc.edu.au/Products/CORBA/MOF/