4.2 An Abstraction-Based Environment

The development and reuse of fixed or adaptable components is an effective mechanism for obtaining software components appropriate to specific needs. However, our actual goal is to build complete customized products whose elements are demonstrably consistent and complete and to do so with minimum effort. An approach to this can be built upon the mechanism of component reuse for derivation not just of software components but of all elements of a software product (e.g., developer documentation, software, verification materials, delivery materials, and user documentation and training materials).

This section describes an early example of an abstraction-based software factory[1]. This factory took the form of a mechanism for specifying and generating a software product based on a generic descriptive model of its elements. The products it could be used to build were implicit to the concepts that could be expressed in that model and the associated component implementations that represented those concepts.

This factory had four elements:

- A characterization of an abstraction that expresses the unifying concept for a set of similar products that can be built with this factory and the associated decisions that a developer must specify in order to derive a particular product
- a collection of "raw materials": abstractions, each corresponding to a family of similar components and expressed in the form of an adaptable component, and a mechanism for deriving instances of each abstraction based on developer-specified decisions
- a conceptual architecture that defines the components needed to build each of the elements of any constructible product and how these components are organized to form each of those elements for a particular product

[1] A prototype of such a factory was developed in 1984-88 as a precursor to the concept of a domain-specific environment (section 4.3). <www.domain-specific.com/PDFfiles/ABE-Spectrum.pdf>

- a process that prescribes how developers, by specifying required decisions and evaluating the resulting product elements in aggregate, can produce an acceptable product

Each of these elements will be illustrated using notational examples of their contribution to the operation of the factory.

Product Specification

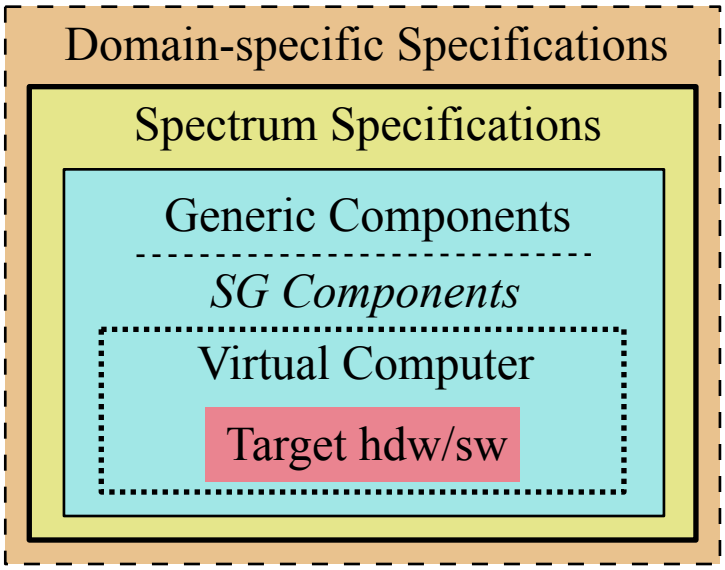

Figure 4.2-1. Levels of Product Specification

A Process for Deriving a Customized Software Product

Figure 4.2-2. DsE Product Manufacturing

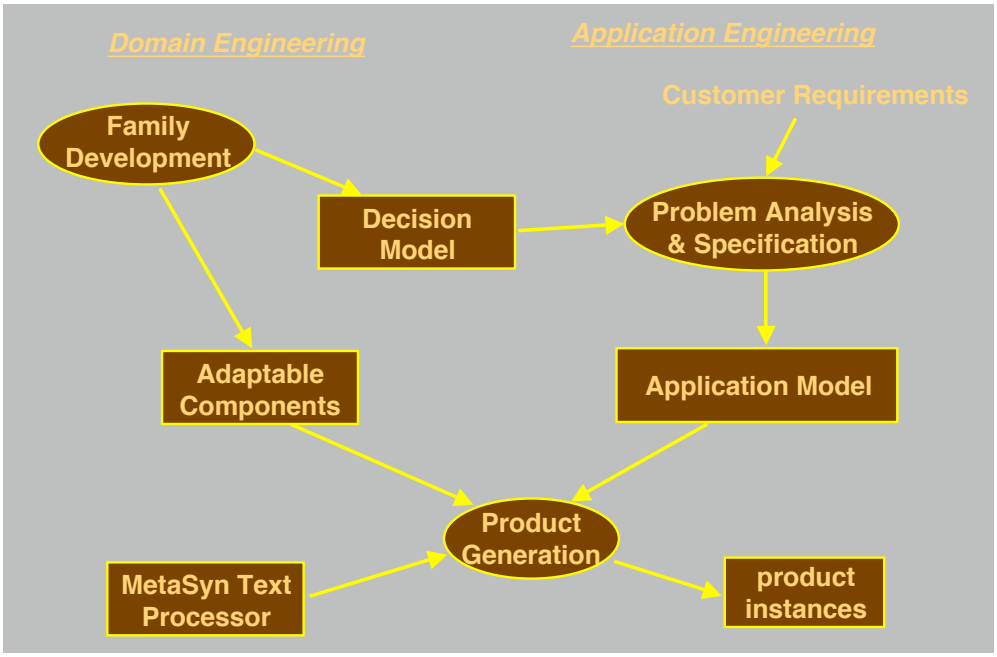

Figure 4.2-2. DsE Process

<figure: concepts of capab avail to products, notional std arch (semantic data model, data deriv strat, devices/intfcs, process control)>

A Generic Abstraction-Based Environment/Software Factory

Briefly, the following describes the construction of Spectrum, an early software product factory. The capabilities of Spectrum were limited in terms of what can be built today and more so what will be possible to build in the future. Spectrum was based on the concept of a generic family of "generic" software products and was envisioned as an abstraction-based environment for which a product was a composition of instances of three generic abstractions:

- virtual hardware
  - a computer (data types, sequential processing, concurrency, communication)
  - a set of logical devices (user displays, printers, data storage, network access)
- software services

  - data abstraction (application data types, semantic data storage)
  - logic abstraction (knowledge representation and inferencing, strategy control)
  - user interfaces (textual documents, structured forms, graphical forms, interaction control)
  - code generation (modules, component instantiation, product configuration)
- product behavior
  - function definition (world model, external interface) {specification, generation, validation of a product)}
  - shared functionality (user state, user assistance, interaction conventions, specification elements)

The hardware and software services abstractions were used to build both applications and Spectrum itself. Lacking an integral focus on a product family, the products that could be built with Spectrum were determined "bottom-up": the capabilities a product could be given depended on the functionality that existing components provided; adding new capabilities required both building new or revising existing abstractions and extending the application behavior abstraction to specify, generate, and validate the associated capabilities for a product.

The Spectrum user interface for specifying an application had two primary elements: a world model and an external interface. The world model was a semantic data model consisting of a set of object classes in a specialization hierarchy, each class having an associated collection for each member object of typed "value" attributes and class-linked "relation" attributes that identified related objects. Class and attribute definitions could have associated knowledge-based strategies for determining or modifying class membership or attribute values, automatically maintaining dependency consistent as related information changed.

The external interface was organized as a set of "logical" devices, each being realized by means of an identified physical device (e.g., user displays, data storage, or network access). A logical network access device provided the means to send and receive

messages corresponding to the content of a specified world model class. Similarly, a logical data storage device provided the means to share data stored in the world model with an external database. These logical devices included a user display device for each of a set of user roles; a user role corresponded to a set of window-based displays whose content presented information represented in the world model. Each display was presented the information content of one or more objects in its designated world model class and other objects accessible via its relation attributes. Information was displayed in any of several user interface formulations (e.g., text-based documents or structured forms). The customized content of each particular display was generated in the application software to reflect the content and structure of the world model information being displayed.

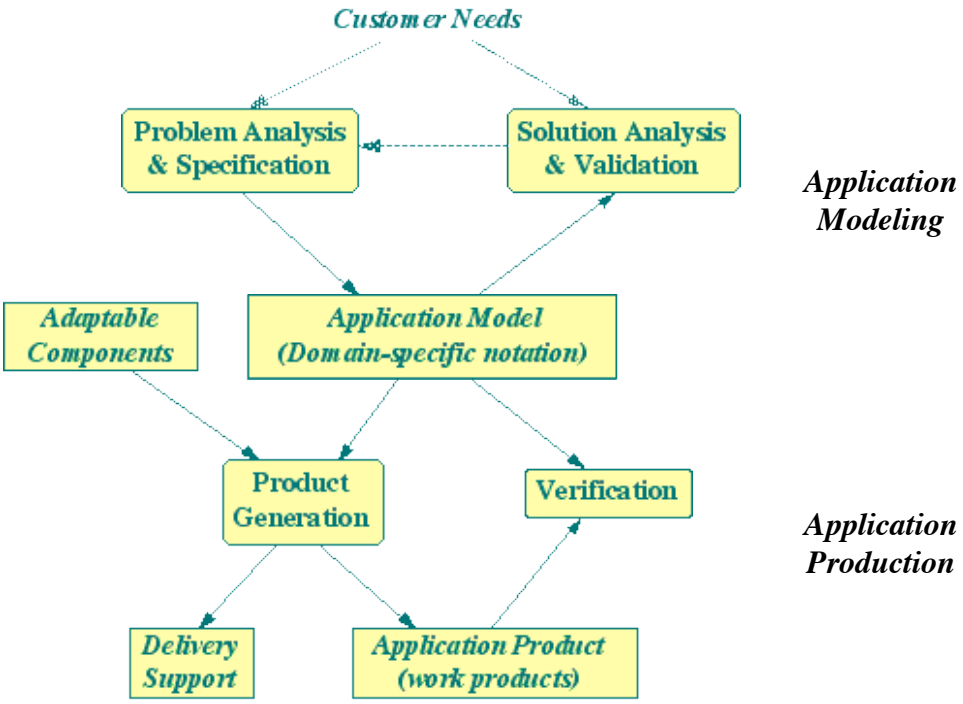

Figure 4.2-: Process for building a software product

In this factory, the product is described from two perspectives: (1) a specification that defines the product's information content corresponding to the knowledge its needs about the static and dynamic nature of the world it is meant to affect and (2) a specification of what mechanisms the product uses to detect the world, informing its

internal dynamic model of that world, in order to determine what actions it will take to effect that world using mechanisms it has available.

World Model Specification

(information space: natural/artificial environment, entities)

<figure: WM conceptual structure>

External Interface Specification

(logical devices/entities: hardware, system, user)

<figure: EI conceptual structure>