October 2001

Software Mass Customization

Charles W. Krueger, Ph.D.

Software mass customization is a powerful business model as well as a powerful engineering model that enables software organizations to efficiently engineer customized variations of their software products. This document describes software mass customization in three parts. First is an overview of software mass customization and its benefits. Second is an introduction to a software engineering technology that enables organizations to quickly adopt software mass customization as part of their business and engineering models. Third is a collection of scenarios of how software product organizations can rapidly adopt software mass customization.

Contact Information:

www.biglever.com
info@biglever.com
(512) 426-2227

1.0 Software Mass Customization and its Benefits

Software mass customization focuses on the *means* of efficiently producing and maintaining multiple similar software products, exploiting what they have in common and managing what varies among them. This is analogous to what is practiced in the automotive industry, where the focus is on creating a single production line, out of which many customized but similar variations of a car model are produced. The powerful, though subtle, essence of this description is the focus on a singular means of production rather than a focus on producing the many individual products. Once the means of mass customization is established, production of the individual products is more a matter of automated instantiation rather than manual creation.

Real world success stories of software mass customization come from diverse areas such as mobile phones, e-commerce software, computer printers, diesel engines, telecom networks, enterprise software, cars, ships, and airplanes. Each of these examples relies on creating and maintaining a collection of similar software systems. For example, a mobile phone company may have tens or hundreds of different phone models, each of which uses software that is unique though similar to software in the other models. New models may be introduced every two weeks. By using software mass customization techniques to exploit what their software systems have in common and to effectively manage the variation, companies are reporting order of magnitude reductions in time-to-market, engineering overhead, error rates, and cost[1,2,3,4,5,6].

What is most interesting from these success stories, however, is that the tactical improvements in software engineering are large enough to have strategic impact on the business of a company. By bringing larger numbers of precisely customized products to market faster and with better quality than their competitors, companies have been able to assume market leadership.

1.1 Challenges of Software Mass Customization

Many of the companies who have reported great success with software mass customization have also reported great challenges and costs in making the move to that model. Adoption times are often measured in terms of years and the costs in millions of dollars. Often, key architects and senior technical personnel must be taken off line for many months to prepare for the move to software mass customization. Often, organizational restructuring and process re-tooling are required. Although many organizations are now learning of the huge potential benefits of software mass customization, the associated costs, risks, and resources are prohibitive barriers for many.

The tension between the potential benefits and imposing challenges of software mass customization is often manifest in the interactions between marketing and engineering groups in a company. Sales and marketing frequently encounter opportunities where customizations of their software products could result in additional revenue. From a business perspective, software mass customization represents a lucrative strategic model for dominating market share, expanding into new market segments, and closing complex deals with demanding customers. However, engineering must frequently resist customization requests because of the associated high level of effort, resources, costs, and risks.

Why is software mass customization more difficult than simply (1) building a single software system, and then (2) building the collection of small variations? Why do we need a major shift to complex and heavyweight software engineering technologies, methods, processes, and techniques?

The answer is that, over the past several decades, we have developed formal tools and techniques for building single software systems (item #1 above), but we have no formal tools or techniques in our arsenal for building and managing a collection of small variations for a software product line (item #2 above). To compensate for this void, software engineers historically have relied on informally contrived solutions such as IFDEFs, configuration files, assembly scripts, install scripts, parallel configuration management branches, and so forth. However, these informal solutions are not scalable; they are not manageable beyond a small number of product variations. Moreover they are code-level mechanisms that are ill-suited to express product-level constraints. More recently, research has focused on some of software engineering’s most powerful and complex solutions for managing product line variation, but these have the associated high cost of adoption.

The current situation, then, can be summarized as follows. Software mass customization has the potential to bring order-of-magnitude improvements to an organization’s performance, but the practices up to this point combine a massive up-front investment at the highest organizational levels with unsatisfactory code-level mechanisms to actually manage the variabilities. The time is right for another approach.

2.0 Simplifying Software Mass Customization

Using one of computer science's most powerful principles, *separation of concerns*, BigLever Software has created *BigLever Software GEARS*, a software mass customization technology that enables organizations to quickly adopt a software mass customization business and engineering model[7,8,9,10,11]. GEARS works in conjunction with the existing tools and techniques, currently used for building single software systems, so that mass customization is a straightforward extension to single system engineering. The separation of concerns is applied so that the technology is independent of programming language, operating system, configuration management system, build system, and so forth. Furthermore, it does not depend on a particular domain modeling language, architecture language, or design language.

By extending the existing single system technology set with a formal technology focused on software mass customization, software organizations can achieve the order of magnitude benefits of software mass customization with an order of magnitude less time and effort than has previously been required. Rather than timeframes of months and years, BigLever talks in terms of what can be accomplished the first day, week, or month.

2.1 BigLever Software GEARS

BigLever Software GEARS[1] provides the *infrastructure* and a *development environment* for creating a *software mass customization production line*. Revisiting the analogy to automotive mass customization, where a single *production line* is used to manufacture many customized variations of a car model, GEARS is analogous to the infrastructure and technology used to create the automotive production facility. That is, GEARS is used to create a single software *production line*, out of which many customized variations of a software system can be produced.

Imagine, for example, that your company has already manually created three different variations of a software product for three different customers or different market segments. Because these product customizations were created under different deadlines, it was easiest to just create and maintain three independent copies of the system in parallel (for example, on different configuration management branches). However, the effort of parallel maintenance of these three system versions is taking its toll, and requirements for more customized variations are looming in the sales and marketing department.

Using GEARS, these three system copies are consolidated into a single software mass customization production line. Software that is common among all three systems is factored out. For software that varies among the three, the GEARS infrastructure is used to encapsulate the differences at the point of variation in the source code, along with the logic descriptions for choosing among the differences at production time. With the software now structured into a single GEARS software production line, the three individual products can be assembled with the push of a button. The production line can be easily extended and maintained to accommodate new customized products, requirements, or defects. Note again that the focus shifts from developing and maintaining three separate products to developing and maintaining a single production line.

1. Patent pending

2.2 The Software Mass Customization Layer

BigLever Software GEARS works at the granularity of files. By not intruding in your files, GEARS remains neutral to programming language, compilers, operating system, editors, and so forth. GEARS works equally well with files that contain source code, documentation, test cases, requirements, and even binary multimedia files.

Figure 1 illustrates where the GEARS software mass customization *layer* fits in, relative to conventional technology layers. At the bottom layer is the operating system's file system. Configuration management extends that layer by providing management for file and system versions that *vary over time*. GEARS extends that layer by providing mass customization of system versions that *vary at a fixed point in time*.

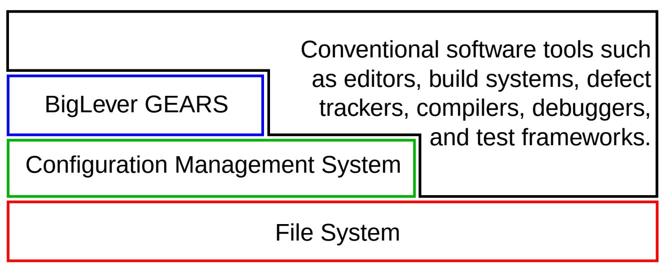

FIGURE 1. The Software Mass Customization Layer of GEARS

2.3 GEARS Infrastructure, Development Environment, and Actuator

GEARS comprises mass customization infrastructure, a mass customization development environment, and a mass customization actuator. The infrastructure structures software into a mass customization production line. The development environment has editors and browsers to view, create, modify, and maintain the production line. The actuator activates the production line to produce the individual software product instances.

The software mass customization infrastructure of GEARS has three major components, *feature declarations*, *product definitions*, and *automata*:

- *Feature declarations* model the scope of variation in the production line. Figure 2 illustrates an editor from the GEARS development environment, viewing a collection of feature declarations in the infrastructure. In this example there are declarations of four features that can vary in the production line, which in this case is an automotive e-commerce web site. *AutoLocator* is a Boolean parameter indicating whether or not an automobile locator feature is available, *ServerCount* indicates how many servers are configured, *Brand* indicates the automotive brand for the site, and *DeliveryOptions* indicates which vehicle delivery options are supported by the site.
- *Product definitions* model the product instances that can be created from the production line. Figure 3 illustrates an editor from the GEARS development environment, viewing a product definition in the infrastructure. Values are selected for the feature declarations in the previous figure, indicating the desired customized features of the product. The product in this example will include the *AutoLocator,* operate on a 10 server configuration, display the Ford brand of vehicles, and support factory and dealer delivery options.
- *Automata* encapsulate source code variants that exist in the production line and the logic for selecting among the variants. Figure 4 illustrates an editor from the GEARS development environment, viewing the selection logic in an automaton. In this example the automaton selects among two file variants, *normal.java* and *stub.java*, depending on the value of the *Boolean* feature declaration *AutoLocator.*

The GEARS actuator is responsible for configuring a product instance from the source files, declarations, definitions, and automata in a production line. For example, if the actuator were applied to the Ford product definition in Figure 3, the automaton logic in Figure 4 would be actuated to select the *normal.java* variant since the *AutoLocator* value is defined as *true*. By actuating all automata in a production line, a complete product is configured.

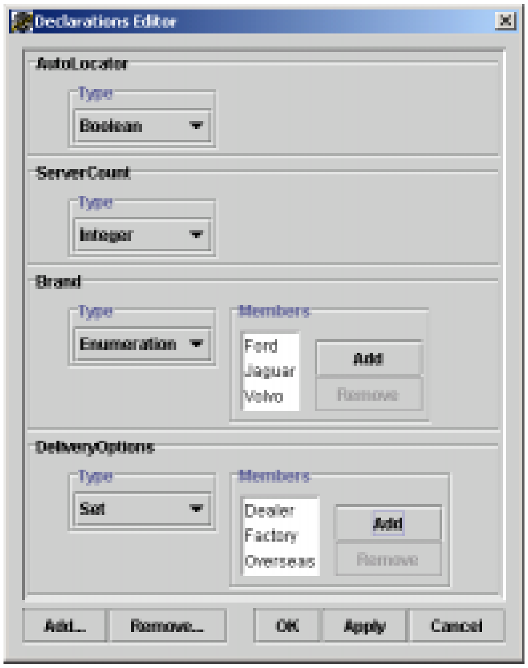

FIGURE 2. GEARS Feature Declarations

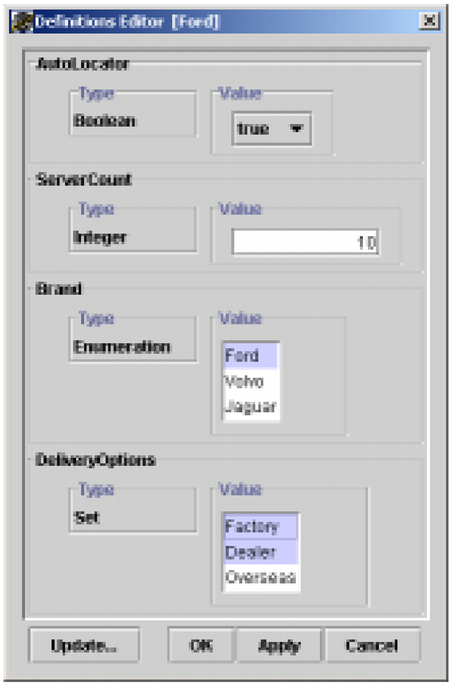

FIGURE 3. GEARS Product Definition

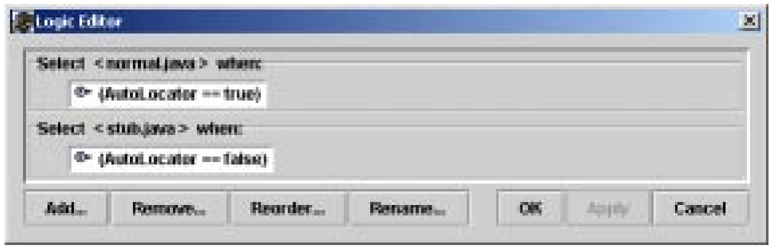

FIGURE 4. GEARS Automaton Logic

3.0 Models for Adopting Software Mass Customization

Organizations adopting software mass customization with BigLever Software GEARS can operate using one of three broad approaches. We have termed these as *proactive*, *reactive*, and *extractive*.

With the *proactive* approach, the organization analyzes, designs and implements a complete software mass customization production line to support the full scope of products needed on the foreseeable horizon. From the analysis and design, a complete set of common and varying source code, feature declarations, product definitions, and automata are implemented. This corresponds to the heavyweight approach discussed earlier in Section 1.1, “Challenges of Software Mass Customization,” on page 2, while at the same time utilizing GEARS as the software mass customization infrastructure and development environment.

With the *reactive* approach, the organization incrementally grows their software mass customization production line when the demand arises for new products or new requirements on existing products. The common and varying source code, along with the feature declarations, product definitions, and automata, are incrementally extended in reaction to new requirements. This incremental approach offers a quicker and less expensive path into software mass customization.

With the *extractive* approach, the organization capitalizes on existing customized software systems by extracting the common and varying source code into a single production line. Using the BigLever GEARS infrastructure, the feature declarations, product definitions, and automata are created as the commonality and variation is identified during the extraction. This high level of software reuse enables an organization to very quickly adopt software mass customization.

Note that these approaches are not necessarily mutually exclusive. For example, a common approach is to bootstrap a software mass customization effort using the extractive approach and then move on to a reactive approach to incrementally evolve the production line over time.

The following sections provide more detail on the extractive, reactive, and proactive approaches to software mass customization using BigLever Software GEARS.

3.1 Extractive

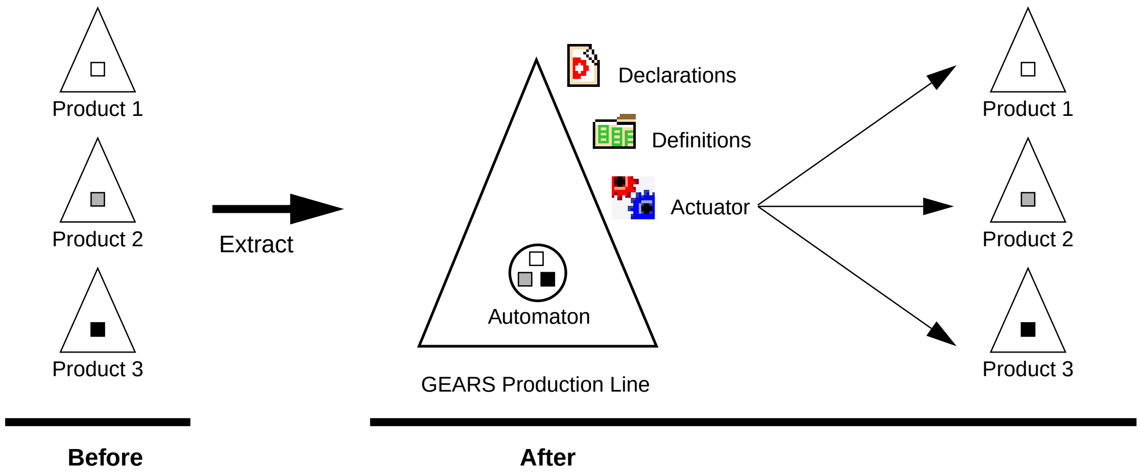

FIGURE 5. Extractive Model of Software Mass Cutomization

The extractive approach to software mass customization is appropriate when you have an existing collection of customized systems that you want to reuse. It is most appropriate when the collection of systems has a significant amount of commonality and also consistent differences among them.

It is not necessary to perform the extraction from all of the pre-existing systems at once. For example, a subset of the high-payoff, actively used systems might be extracted initially and then the remainder incrementally extracted as needed.

The high level tasks for the extractive approach are as follows:

1. Identify commonality and variation in the existing systems
2. Factor into a single BigLever GEARS production line
   - create a single copy of the common software
   - create feature declarations that model the scope of variation among the existing systems
   - encapsulate variation points into automata
   - program the automata logic to map declaration parameter values to variant selections in the automata
   - create the product definitions for the desired product instances by selecting values for each of the feature definition parameters

After the production line has been populated, product instances are created (with the push of a button) as needed via the actuator. Software mass customization now becomes the mode of operation as focus shifts to maintaining and enhancing the single production line.

3.2 Reactive

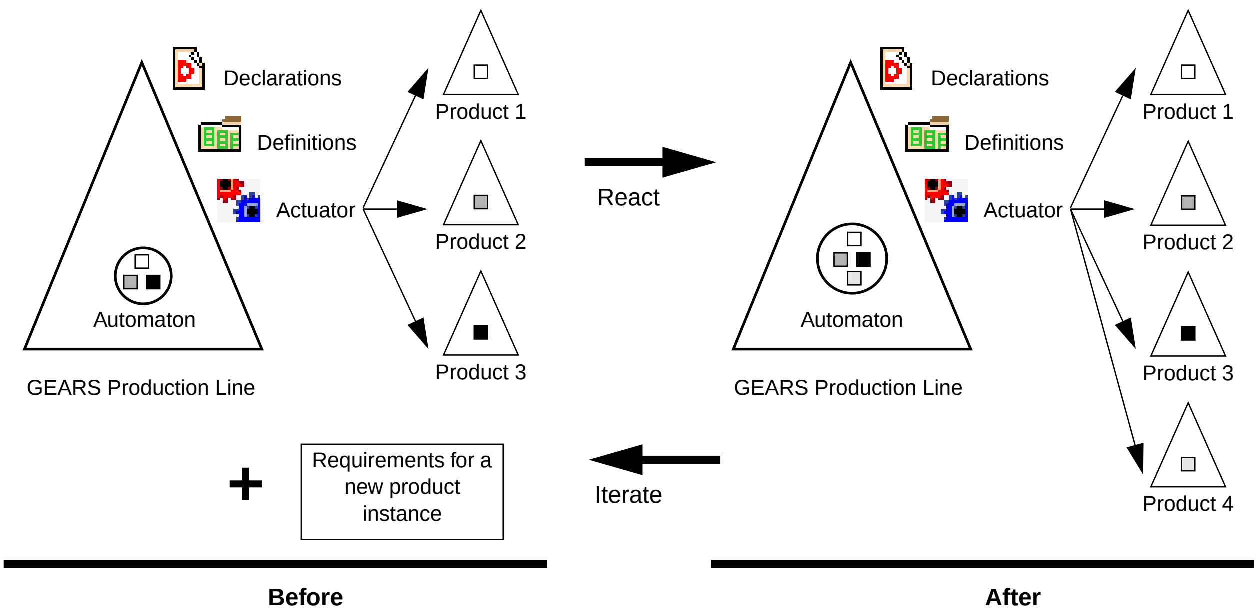

FIGURE 6. Reactive Model of Software Mass Cutomization

The reactive approach to software mass customization is an incremental approach. It is appropriate when the requirements for new products in the production line are somewhat unpredictable. For example, when unexpected requirements from new customers are common, the reactive approach is appropriate. The reactive approach allows for rapid adoption of mass customization since a minimum number of products must be incorporated in advance.

The high level tasks for incrementally adding a new product using the reactive approach are:

1. Characterize the requirements for the new product relative to what is currently supported in the production line
2. It is possible that the new product is currently within the scope of the current production line. If so, skip to step 4.
3. If the new product is not in scope, perform the “delta engineering” to the production line on any or all of the declarations, automata, common software, and definitions in order to extend the scope to include the new requirements
4. Create the product definition for the new product by selecting values for each of the feature declaration parameters

3.3 Proactive

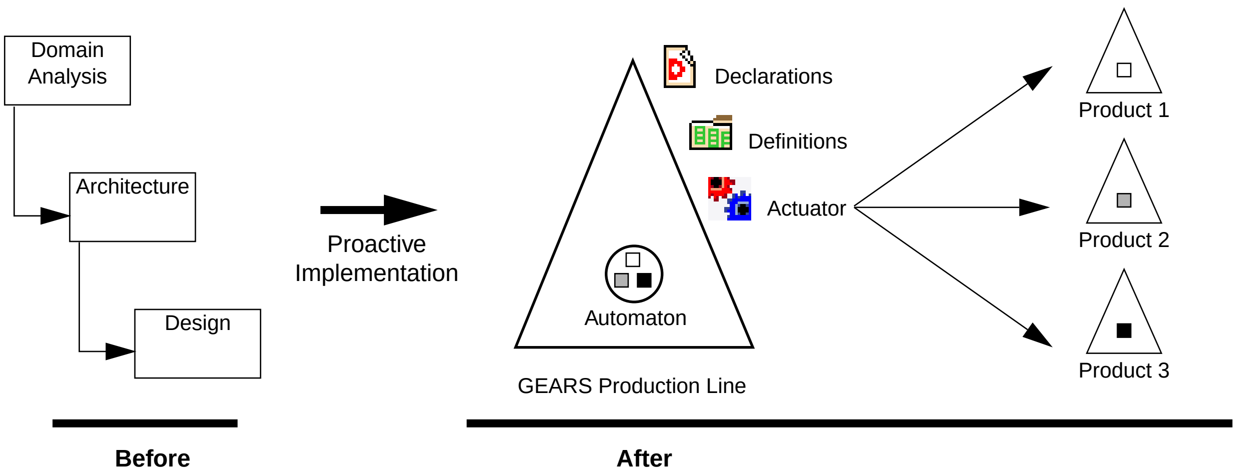

FIGURE 7. Proactive Model of Software Mass Cutomization

The proactive approach to software mass customization is similar to the waterfall approach for single systems. It is appropriate when the requirements for the set of products needed, extending to the far horizon, are well defined and stable. The proactive approach requires considerably more effort up front, but this drops sharply once the production line is complete. If the up front cost is prohibitive or if the risk of guessing wrong is high, consider the reactive approach.

The high level tasks for the proactive approach are as follows:

1. Perform domain analysis and scoping to identify the variation to be supported in the production line
2. Model the product line architecture to support all products in the production line
3. Design the common and variant parts of the system
4. Finally, implement the common and variant parts of the system using the BigLever GEARS declarations, definitions, automata and common source.

Once the production line has been implemented, all that remains is to create product instances (again, with only the push of a button) as needed via the actuator. With the proactive approach, if new products are needed, most likely they are within the existing scope and can be created by simply adding a new product definition in the GEARS infrastructure. Maintenance and evolution are performed directly on the single production line.

4.0 Example: Globalization & Localization using Software Mass Customization

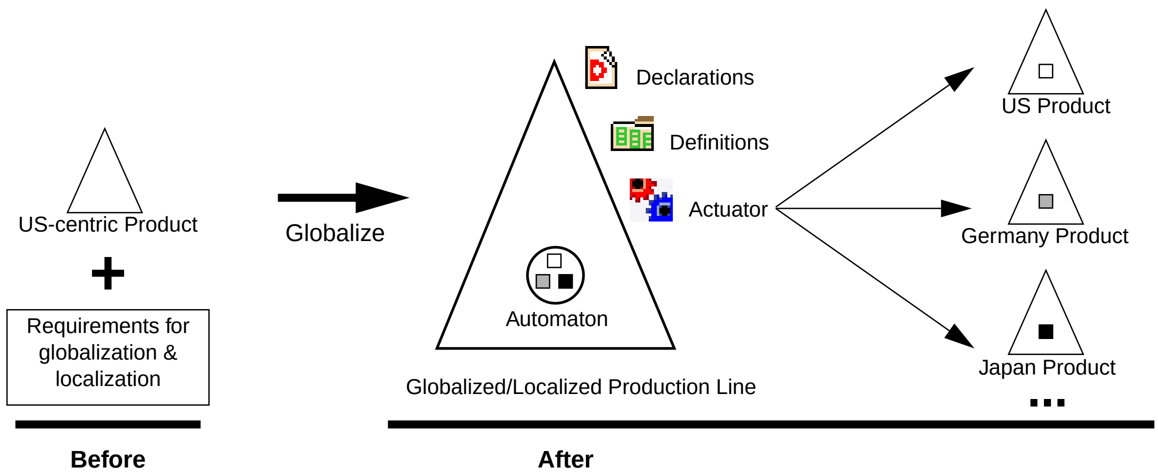

FIGURE 8. Globalization/Localization Production Line

This section illustrates how to use software mass customization for globalization and localization, or internationalization, of a US-centric software system. The reactive approach with BigLever Software GEARS, as described in Section 3.2 on page 9, is particularly adept at this task since it does not assume that the US-centric product was designed or implemented with globalization in mind.

4.1 Step 1. Initializing the Production Line

Comparing the "Before" state of Figure 8 with that of Figure 6, the US-centric product can be viewed as a production line with no variants. That is, US-centric product can serve as the initial basis for the production line.

From BigLever GEARS, we select the "Create New Production Line" operation and point to the root of the existing US-centric product source tree. GEARS creates the initial infrastructure for the production line and creates empty declarations and definitions. The US product is now a GEARS production line.

4.2 Step 2: Declaring the Globalization Model

The key globalization dimensions for the production line are declared next. These may come from a combination of industry standards, company standards, and from an analysis that is specific to this product and its target customer base.

Figure 2 illustrated the GEARS editor for declaring dimensions of variation. Typically a system architect or someone in a similar role is responsible for creating and maintaining the declarations for a production line. In product line engineering terms, this role is defined as a *domain engineer.*

Globalization variables that might be declared for this example include:

- an Enumeration of countries, from which exactly one will be selected for a product instance
- a Set of religions that potential users may belong to. When a product is defined, a subset will be selected from the religion Set.
- a Boolean variable indicating whether to use Euros or local currency. This variable is used in conjunction with the country variable since it is only applicable for some European countries.
- an Integer variable that expresses a taxation code that is dictated by the accounting department
- an Enumeration or String that models the language and/or dialect used for textual display
- a Character to model diplomatic status or immunity of the user group
- an Enumeration to model the citizenship of the users, such as US citizens working in a foreign country

4.3 Step 3: Defining the Localized Product Instances

If the initial target collection of localized products is known up front, these can be defined next using the product definition editor that was shown in Figure 3. A named product definition is created for each product instance that we want to instantiate from the production line.

The task of defining a product simply consists of selecting a value for each of the feature declarations from the previous step. Figure 2 and Figure 3 clearly demonstrate this relationship.

The task of creating a product definition is typically carried out by a system architect or similar role. In product line engineering terms, this role is defined as an *application engineer.*

With GEARS, the *domain engineer* that creates declarations and the *application engineer* that create definitions is likely to be the same person since the tasks are relatively simple and closely related.

It is interesting to note from this example that, in software mass customization terms, globalization corresponds to domain engineering and localization corresponds to application engineering.

4.4 Step 4: Encapsulating Globalization and Localization points in the source code

Next it is time for the developers to work their magic. Delving into the source base of the US-centric product, they identify US-specific areas in the source code that must be generalized to support other locales.

The files that need to be globalized are converted from common files to automata using the GEARS development environment. For example, a timesheet form in a US application may have an overflow area to compute overtime hours worked, whereas in Germany this same form must provide a warning when the legal maximum number of hours worked during a week is being approached rather than overflowing into overtime. These two timesheet variants would be encapsulated in a single timesheet automaton. The logic description in this automaton is then programmed to select among the US and German file variants using the declaration variables and values. Figure 4 illustrates selection in the automaton logic.

4.5 Step 5: Actuating the Production Line to create the Product Instances

After the declarations, definitions, and automata have been created to form a complete software mass customization production line, the localized product instances for different regions can be produced. The actuation operation of GEARS takes a product definition as input and creates a corresponding product instance as output.

4.6 Maintaining and enhancing the Production Line

After the production line is established, all maintenance and evolution is performed on the single production line rather than the individual products. For example, a bug fixed once in the common code is fixed for all product instances.

As requirements for new locales are introduced, the declarations, definitions, and automata are incrementally extended as necessary. The entire production line evolves as a whole in configuration management, so that we can always go to the production line state from two months ago in order to re-actuate and reproduce any product that was shipped at that time.

References

1. Software Engineering Institute. *The Product Line Practice (PLP) Initiative*, Carnegie Mellon University, www.sei.cmu.edu/activities/plp/plp_init.html
2. Weiss, D., Lai, R. 1999. *Software Product-line Engineering*. Addison-Wesley, Reading, MA.
3. Clements, P. and Northrop, L. 2001. *Software Product Lines: Practices and Patterns*, Addison-Wesley, Reading, MA.
4. Bass, L., Clements, P., and Kazman, R. 1998. *Software Architecture in Practice*. Addison-Wesley, Reading, MA.
5. Jacobson, I., Gris, M., Jonsson, P. 1997. *Software Reuse: Architecture, Process and Organization for Business Success*, ACM Press / Addison-Wesley, New York, NY.
6. *Software Product Lines. Experience and Research Directions. Proceedings of the First Software Product Lines Conference (SPLC1)*. August 2000. Denver, Colorado. Kluwer Academic Publishers, Boston, MA.
7. BigLever Software, Inc. Austin, TX. *www.biglever.com*
8. Krueger, C. Using Separation of Concerns to Simplify Software Product Family Engineering. *Proceedings of the Dagstuhl Seminar No. 01161: Product Family Development.* April 2001. Wadern, Germany.
9. Krueger, C. Easing the Transition to Software Mass Customization. *Proceedings of the 4th International Workshop on Product Family Engineering*. October 2001. Bilbao, Spain. Springer-Verlag, New York, NY.
10. Krueger, C. Software Reuse. 1992. *ACM Computing Surveys*. 24, 2 (June), 131-183.
11. Krueger, C. 1997. *Modeling and Simulating a Software Architecture Design Space*. Ph.D. thesis. CMU-CS-97-158, Carnegie Mellon University, Pittsburgh, PA.