Active Software Artifacts

Mehmet Aksit¹, Francesco Marcelloni², Bedir Tekinerdogan¹,
Klaas van den Berg¹ and Pim van den Broek¹

¹TRESE project, Department of Computer Science,
Centre for Telematics and Information Technology, University of Twente,
P.O. Box 217, 7500 AE, Enschede, The Netherlands.
email: {aksit | bedir | vdberg | pimvdb }@cs.utwente.nl
www server: <http://www.trese.cs.utwente.nl>

²Department of Information Engineering, University of Pisa,
Via Diotisalvi 2, 56156 Pisa, Italy
email: france@iet.unipi.it

Abstract. There are many similarities between industrial goods manufacturing and software development processes. This paper first briefly analyzes the recent developments in goods manufacturing, and then identifies the equivalent techniques in software technology. It is claimed that products developed during software manufacturing must be modeled as active artifacts. As a possible approach in this direction, an object-oriented artifact production framework is presented and evaluated.

1. Manufacturing Techniques and Software Engineering

Despite of all the efforts made, developing cost-effective software systems remains a difficult task. There are many similarities between industrial goods manufacturing and software development processes, and therefore it may be worthwhile to compare these disciplines together. The progress made in industrial goods manufacturing can be roughly classified as mechanized and computerized techniques.

The mechanized manufacturing age started through the introduction of production lines. Compared to the early techniques, the production lines approach has increased the efficiency of manufacturing incomparably because production facilities could be shared, managed and automated. In this approach, however, to be able to have a cost-effective production, goods have to belong to the same product line, require similar production facilities, and the number of manufactured goods must be large enough.

During the last decade, software engineering research activities have shown many similarities with the mechanized goods manufacturing techniques. For example, to be able to specify software product lines, software architecture specification techniques have been developed. Software development methods have allowed software engineers to share the same notation, rules and processes. Component-based development techniques have enabled standard software parts to be developed and reused. Due to continuous increase in complexity of software systems, however, it

looks like those goals in cost-effective software development remain equally challenging like before.

Let us now consider computerized goods manufacturing techniques, which have been introduced to deal with similar problems like we face today in software engineering. Firstly, computer aided design environments have been developed with increasingly high-level semantic support, such as calculating the mechanical properties of the materials used. In addition, a substantial progress has been made in design automation techniques. Secondly, to be able to increase the flexibility of the production processes, computer integrated manufacturing techniques have been developed. These aim at smooth integration of various phases in the manufacturing process. Finally, to simplify the maintenance activities, design and production information have been captured and included in the delivered products as built-in information so that service engineers can retrieve this information whenever necessary.

These recent developments in goods manufacturing imply the following equivalent techniques for software industry: computer-aided design of software systems and design automation, computer-integrated manufacturing of software systems and integrating design and manufacturing information with software products.

To realize computerized software manufacturing processes, first of all, every generated product, such as architecture specification, analysis and design models, design decisions, documentation and software components must be considered as an artifact. This is necessary for reasoning about software artifacts and their manufacturing processes. Secondly, these artifacts have to be designed as *intelligent* software modules. Being intelligent means that every artifact defines its own rules for its creation, definition of parts, quality control and coordination with other artifacts. If artifacts are designed as intelligent software modules, they may actively assist software engineers. Thirdly, artifacts must record their context and interdependencies. This helps software engineers walk through the related artifacts, for example, from executable software modules to architecture specification artifacts. Finally, because of the complexity of the problem, artifacts must be extensible and use open-ended protocols so that new artifacts can be introduced whenever necessary.

2. The Artifact Production Framework

To create reusable artifacts and open-ended protocols, we adopted object-oriented modeling principles. For uniformity, each artifact in production is derived from the super class *ArtifactInProduction*. As shown by Figure 1, class *ArtifactInProduction* is composed of two interacting parts: *Producer* and *Controller*. The part *Producer* carries out the artifact production process, and the part *Controller* is responsible for the quality of the artifact.

Fig. 1. Structure of an artifact in the production process.

To initiate the production process, the corresponding Input Provider supplies the pre-artifacts. Pre-artifacts are the necessary sub-products to produce an artifact. Once the pre-artifacts are supplied, the necessary information is retrieved from them. The subpart Input Retrieval describes the sequence of actions needed to implement the retrieve protocol. For example, assume that the software engineer is identifying a *class*. This requires a pre-artifact of type Entity-in-a-Requirement-Specification. To be able to reason about a class, the *relevance* and *autonomy* values of the supplied entity must be retrieved.

The subpart Production Specification defines a set of production rules which describe how the retrieved information is transformed into an artifact. For example, if an entity is *relevant* and *autonomous*, then the corresponding class can be identified. Further, the subpart Production Specification defines rules to collect the quality measures. The part Controller compares the quality measures to the reference measures and generates an error signal. This signal is examined by the subpart Quality Policy to decide which action to take. We consider here three possible actions: raise an exception (when the error is too high), improve (when the error is acceptable, but should be lowered) and commit (when the error is negligible). The exception message is captured by an external controller, which can be a software engineer or a higher-level control subsystem. The messages *improve* and *commit* are processed by the subpart Actuator. If the message is *improve* then Actuator requests the *input providers* to improve the quality of the information, whereas in case of *commit* the produced artifact is delivered. Notice that complex production systems can be defined by creating chains of artifacts in production.

3. Evaluation and Conclusions

In section 1 it was stated that every generated artifact has to be defined as an intelligent software module. The artifact production framework shown in Figure 1 is designed as a generic active object with its own creation, quality control and interaction rules. These rules are preprogrammed based on the type of artifact to be produced. However, each artifact shares the same structure and external protocols. This experimental framework has been designed and built using the Smalltalk language [1]. Most adopted rules are expressed using fuzzy-logic [4]. This was found necessary to model the design heuristics more precisely than two-valued logic based rules. To verify the framework, we are currently defining artifacts for some popular methods such as OMT [5].

To be able to integrate the design information and context, each artifact stores the values used in the production and control processes. Further, interdependencies among artifacts are preserved. We are developing some tools to trace the design decisions as well as interdependencies among artifacts. For example, starting from the final object models, it should be possible to trace all the related design decisions up to the requirement specification level. Our current experimentation has been limited to some simple cases and design rules. We have initiated a pilot project to apply the framework in an industrial application called Integrated New European Car Dealer Information System [2].

Similar approaches have been proposed in the literature to cope with the complexity of software production. For example, at the Software Engineering Institute, a research activity on product-line systems has been established [6]. Further, there are active groups carrying out research activities on the so-called Automated Software Engineering [3]. However, to the best of our knowledge, fuzzy-logic techniques implemented in an active object-oriented framework have not been realized before.

References

1. J.B.W van Dongen, An Object-Oriented Production Framework for Software Development, M.Sc. Thesis, University of Twente, June 1997
2. INEDIS project description, University of Twente, 1996.
3. Journal on Automated Software Engineering, Kluwer Academic Publishers, ISSN 0928-8910.
4. F. Marcelloni and M. Aksit. Applying Fuzzy Logic Techniques in Object-Oriented Software Development, to be published in this volume.
5. J. Rumbaugh et al. Object-Oriented Modeling and Design, Prentice Hall, 1991
6. Software Engineering Institute, Product-Line Systems, http://www.sei.cmu.edu/technology/product_line_systems/