Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 27 (2015) 198 – 201

www.elsevier.com/locate/procedia

13th CIRP conference on Computer Aided Tolerancing

Construct surface characterization system by assembling functional components dynamically

Xiangqi Lan*, Xiangqian Jiang, Wenhan Zeng, Liam Blunt

EPSRC Centre for Innovative Manufacturing in Advanced Metrology,
School of Computing and Engineering, University of Huddersfield, Huddersfield, HD1 3DH, UK

* Corresponding author. Tel.: +44-01484-473538; fax: +44-01484-472161. *E-mail address:* XiangqiLan@gmail.com

Abstract

Surface characterization of manufactured components is regarded as an important process to figure out surface features, which are closely related to the manufacture process and will affect their functionality. Due to the complicated computation, the actual operations are mostly completed by the aid of surface characterization software. Nowadays, these systems are mainly exploited by instrument companies and embedded in surface measurement instruments. Although it is convenient for users to evaluate surfaces straightforwardly after measurement, the results are usually incomparable with those from other surface instruments because of the different characterization systems. Moreover, the system evolution will cost too much due to the lack of flexibility and extendibility. This paper presents a component based architecture which facilitates the system construction by assembling functional components dynamically.

Peer-review under responsibility of the organizing committee of 13th CIRP conference on Computer Aided Tolerancing

Keywords: Surface characterisation; component-oriented development; dynamic configuration

1. Introduction

Surface texture and its measurement are becoming more and more critical and important in the field of high precision engineering and nano-technology. Surface roughness is an important factor in determining the satisfactory performance of a workpiece, for example in tribology and coatings. Also it has been found to be useful in machine tool monitoring [1].

It is well known that surface metrology is a rapidly developing discipline [2]. Although many analysis algorithms and methods have already been specified in ISO standards, there are still some drawbacks to the present characterization techniques with many being only appropriate for some surfaces under certain conditions. Therefore, as a software system in a research field, surface characterization system needs to be updated constantly with the emergence of new algorithms and methods. In contrast to the algorithms of a surface characterization system, the system architecture is of equal importance. As function modules in current surface characterization system are tightly coupled together, it is not conducive to the reuse of function modules and innovation of overall system. A lot of redundant and duplicate works have been done in present characterization systems, and they will be done again when building a new characterization system. It is clearly advantageous to develop a flexible system architecture that can bring huge benefits for surface characterization systems by easily facilitating additional functionality.

This paper proposed a new way to establish the surface characterization system with the improved flexibility and extendibility. Instead of being created as a constant chunk, the system will be constructed by assembling its functional components at runtime. Using this approach, system maintenance and extension becomes much easier. Modifications or changes can be completed inside the component itself without affecting other parts of the system, thus the deficient or redundant component can be easily removed from the system and new components can be

Peer-review under responsibility of the organizing committee of 13th CIRP conference on Computer Aided Tolerancing
doi:10.1016/j.procir.2015.04.066

amended without difficulty. To implement a completely flexible system, Object-oriented development method is insufficient. As the successor of object-oriented software development, component based development is adopted here to realize such a flexible surface characterization system.

2. Component-Based Development

The component is a language-neutral, independently implemented package of software services, delivered in an encapsulated and replaceable container, accessed via one or more published interfaces as shown in Fig. 1. It is not platform constrained or application bound [3].

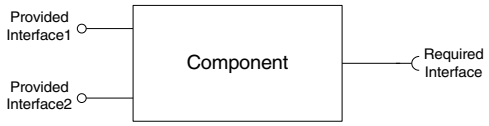

Fig. 1. A component with provided and required interfaces

Component-based software development can greatly improve the development efficiency and ease the extensibility and maintenance of large engineering software. Component-based software applications are composed from diverse software components. Developers and sometimes end-users compose applications from often stand-alone components in flexible ways to achieve a desired set of functions. Two key aims of component technologies are to increase reusability of software in diverse situations without code modifications, and to enable end users to extend and reconfigure their applications via plug and play of components. The framework for CBA (Component-Based Architecture) consists of three major parts: the facilities and services, the application components, and component managers [4]. All the functions are separated from the overall system, and implemented in the components. The system is more like a container where each component can run; it takes charge of the construction, invocation and destruction of components. The event and message map are processed by the system.

The component is an isolated element, a member of an external distributed composite system, and it interacts with the system framework through a set of standard interfaces. Components should be designed to be independent units, much like Lego building blocks. They can be easily added to the system, and existing components may be detached and plugged into other systems. The services offered by components are made public by publishing their interfaces and contracts.

The interface is provided for components to enable asynchronous, dynamic, and anonymous communications. It provides proper connectivity between components and ensures communication between components. This is crucial to implement the dynamic connection of components rather than a statically chained function call. The interface is not only the bridge that connects the components and client; it also illustrates the functions, while the component is the function implementation. The connector is transparent to the client who does not need knowledge of the implementation of components.

The infrastructure for CBD (Component-Based Development) needs three main elements: uniform design notation, standard interfaces, and repositories [4]. The uniform design notation ensures a consistent architectural diagram to describe a component's functions and properties. This is critical to design the collaboration between components and to ease communication between developers. A standard interface for components allows applications in any language on any platform to access their functions. This is achieved by an application in binding to the component model or IDL (Interface Definition Language). IDL is a specification language used to describe a software component's interface. IDL describes an interface in a language-neutral way, enabling communication between software components that do not share the same programming language. Repositories provide the runtime environment for components. Although a component is executable, it cannot run individually. The construction, operation and deconstruction are also managed by the repositories.

3. System Architecture Design

3.1. Separation of Analysis Functions and System Framework

Generally whether a software system is suitable for a certain application is determined by its functions. System functions are of vital importance and always the concern of the users, while the infrastructure is of little interest to the end user. In a surface characterization system, analysis functions such as fitting, filtering and parameters calculation are more emphasized in comparison to the design of system architecture. Therefore, most of present surface characterization systems are equipped with abundant analysis algorithms but in poor system structures. Functions are tightly coupled with other functions or system framework, and it is extremely difficult to maintain the whole system owing to these complex dependent relationships between functions and system framework.

In fact, system architecture is much more crucial for research software systems because most attributes such as the stability, maintainability and extensibility of a software system are closely related to its architecture rather than system functions [5]. However, it is impossible to design system architecture without considering relevant system functions, since system functions are always embedded in system architecture and bound to the system framework. Hence, to design and implement a flexible system architecture is to separate system functions which are prone to change from the system framework physically [3].

3.2. Surface characterization system architecture

According to the principle that the system should be constructed instead of be created from scratch. System

functions should be implemented as components, and the global system consists of these components which are standalone and executable entities. Generally, data importing, fitting and filtering (or any other analysis), parameters calculation and analysis reporting are essential tasks for a surface characterization system. They are the representation of surface verification operations that are used within the surface characterization process. The surface data flow within an ideal surface characterization system is shown in Fig. 2.

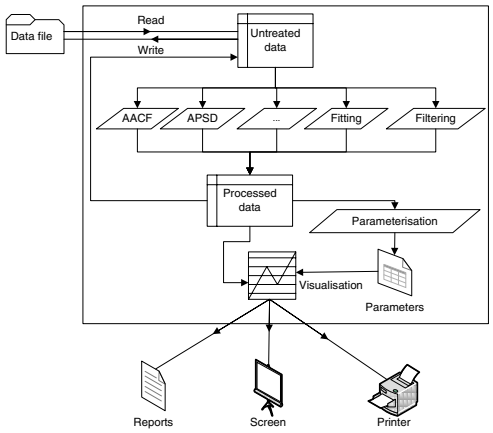

Fig. 2. Data flow within a surface characterisation system

After the measurement, surface data is initially stored in a data file. When evaluating a surface, surface data must be imported to the surface characterization system from a specific file. The surface data was stored in system memory as "untreated data". Once imported, users can select certain operations to process the untreated data according to the specific requirements. As most of the output of these operations are still surface data (though in a modified form) with the same format as the "untreated data", these operations steps can be employed repeatedly. Following surface data treatment, the "processed data" which is expected to include the intended features such as surface roughness can be displayed or parameterized. Finally, the analysis results can be exported to screen, file reports, printer, etc.

3.3. Categorization of system functional components

In the proposed surface characterization system, there are three major types of system functions: data accessing, data processing and data displaying [6]. They comprise the foundation of various data analysis chains. However, they have different I/O properties and cannot be treated in a unified way. It is required to classify them into different types, so that system framework can use a unique method to invoke the functional components with the same type. Furthermore, the categorization of system functional components is the prerequisite of the interface design, because interfaces are the only bridge between them and the system framework.

- Data access components

Data access component implements system I/O functions. These components provide the raw data for the system and can acquire the measurement data from instruments in a direct or indirect way. If a data access component directly acquires measurement data from a particular instrument, it means that this component is a customized component which is usually bound to the instrument and cannot be used by other instruments. For the sake of good reusability, the indirect way to acquire measurement data is considered. The measurement data file is the bridge between instruments and data access components. As illustrated in Fig. 3, a measurement file is the input of data access component, while a data matrix is output for further analysis.

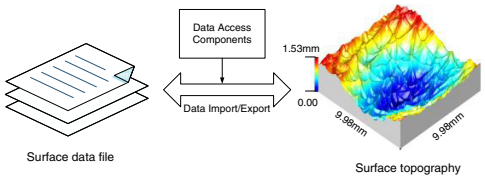

Fig. 3. Function chart of Data Access Components

- Data process Components

Surface analysis and characterization is a sequential procedure of several operations. Every operation in the chain of an operator, such as fitting, filtering and parameter calculation, can be encapsulated as a data process component. Some data manipulations such as data transformation and data arithmetic also belong to data process components. Data process components are a "black box" which is invisible for clients who use this component. As importing data to be processed, a data process component carry out certain operation on the input data and then output the processed data. What is the input and output data are much more critical than how to do the operation. Fig. 4 is an example of Robust Gaussian filtering component which extracts the high frequency band of original surface data.

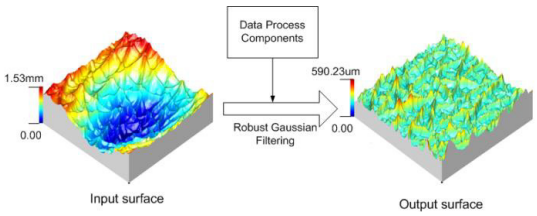

Fig. 4. Function chat of Data Process Components

- Data Display Components

Although the data display component seems to have no relationship with surface characterization process, it is helpful to have an intuitive sense of surface data and explicit understanding of each analysis procedure. Both the input data and output data of each process component need to be displayed by certain display component. Fortunately, three

dimensional graphics display is no longer a barrier as the development of computer graphical technology. Both OpenGL and DirectX can provide a perfect view of real objects, and many implementation details have been omitted. Fig. 5 is the interface of a 3D display component which is implemented by utilizing OpenGL technology.

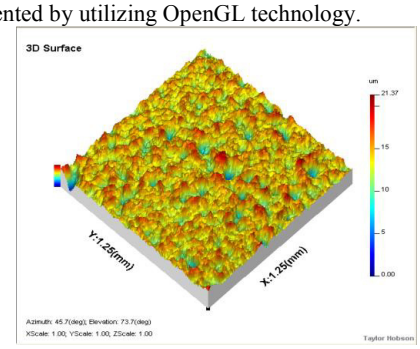

Fig. 5. Data Display Components—3D Display Component

4. Dynamical assembling

In the flexible structure, interfaces acts as a common protocol that every part of the system has to comply with, and they indicate all the interactions among those independent parts. System functions can be recognized by the system framework via their interfaces [7]. Hence, the system becomes much easier to maintain and extend. Any modifications within one part will not affect other parts of the software system. Moreover, it is possible to add new function elements without any change to the original system even when it is running.

The invocation of system functional components is performed according to their interfaces which are determined by their types. In the proposed system architecture, there are three types of system functional components, and they are invoked in different ways within the system framework. System functional components of the same type have to implement the same interface so that they can be invoked in the same way. Among these interfaces, *SSObject* is the foundational interface for the whole system, and every functional component has to realize this interface otherwise it cannot be added to system environment and be executed correctly. The *SSObject* interface is defined as below:

```
//Interface of all external function components
  public interface SSObject
  {
    bool OnRegister();
    bool UnRegister();
  }
```

The first method *OnRegister* is to register some basic component attributes, such as Name, Type, Path and so forth to the configuration database. This happens when adding a new system functional component to the system. Similarly, another dual method *UnRegister* is used to deregister functional components when the functional component is removed from the characterization system.

In addition, every functional component has to realize its own interface. Otherwise, it cannot be recognized as the correct type of functional components. To sum up the proposed system framework can invoke a functional component via its interfaces at any time. The components are absolutely separated from each other. Obviously, it is allowable to change the quantity of system functional components without altering the system architecture and affecting other parts. It also implies that there is no need to rebuild and redeployment whole system when adding or removing system functions and it is desirable for the proposed expandable system. Meanwhile, the whole system is divided into many small individual parts which are much easier to maintain than a single large software system.

5. Conclusion

Surface characterization technology is undergoing tremendous innovation and various creative methods are being employed to evaluate surfaces. A traditional way to extend current characterization system is to integrate new methods to it with system modifications. As a consequence, the whole system becomes increasingly large and complex. Components based development technology aims to construct software systems by gluing loosely coupled components. The complexity of whole system is determined by component types, rather than component quantities. In this characterization system, file formats, analysis algorithms and display methods are three mutable system functions. Therefore, three kinds of components are classified to implement these functions respectively. These components are recognized as additional parts of whole system, and they can be added, substituted or removed without affecting other parts. Besides system developers, any end users also have the right to extend system functions. They can develop their own function components according to their specific requirement as long as such components have implemented pre-defined interfaces.

References

[1] Whitehouse, D.J., Surface metrology. Measurement Science and Technology, 1997. 8(9): p. 955–972.
[2] Jiang, X., et al., Paradigm shifts in surface metrology. Part I. Historical philosophy. Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science, 2007. 463(2085): p. 2049-2070.
[3] McInnis, K. and S. Technologist, Component-based development. The concepts, technology and methodology. Castek Software Factory, 2000.
[4] McArthur, K., H. Saiedian, and M. Zand, An evaluation of the impact of component-based architectures on software reusability. Information and Software Technology, 2002. 44(6): p. 351-359.
[5] Shaw, M. and D. Garlan, Software architecture: perspectives on an emerging discipline. 1996.
[6] X. Lan, X. Jiang., L. Blunt, and S. Xiao, Building a Flexible Surface Characterisation System Architecture.
[7] Ferron, J.R., et al. A flexible software architecture for tokamak discharge control systems. 1995.