Development of an AVM System Implementation Framework

Min-Hsiung Hung*, Senior Member, IEEE, Chun-Fang Chen1, Student Member, IEEE, Hsien-Cheng Huang2, Student Member, IEEE, Haw-Ching Yang3, Member, IEEE, and Fan-Tien Cheng2, Fellow, IEEE

Abstract—Automatic virtual metrology (AVM) is the highest-level technology for VM applications from the perspective of automation. It is an enabler for fast applying VM to all pieces of equipment in a factory. The existing VM-related literature mainly focuses on creating VM models for manufacturing processes using different algorithms or methods and illustrating the defect-detection capability or the VM conjecture accuracy. Only a few of them mentioned how to implement the AVM system, but with limited details. This paper aims to present the development of an AVM system implementation framework, called AVMSIF, to fill this gap. The proposed AVMSIF, together with the developed server-creation approach and XML-based system-operational mechanisms, can allow the complex AVM system to be created in a systematic and easy manner. Also, by adopting plug-and-play interfaces and desired functional modules, the AVMSIF can be applied to different types of equipment in the factory-wide VM deployment. According to the proposed AVMSIF, an AVM system has been successfully created and deployed in our cooperative TFT-LCD factory to confirm the effectiveness of the proposed AVMSIF. The research results of this paper can be a useful reference for high-tech industries, such as Semiconductor and TFT-LCD industries, to construct their AVM systems.

Index Terms—Automatic virtual metrology (AVM), AVM system implementation framework, plug-and-play interfaces, factory-wide VM deployment, Semiconductor and TFT-LCD industries.

I. INTRODUCTION

In Semiconductor manufacturing processes, production equipment needs to be monitored online to assure stable wafer fabrication and high yield rate [1]. Traditionally, equipment monitoring is performed via periodically measuring the monitor wafers that are regularly added in production equipment and processed with production wafers. Since monitor wafers cannot fully represent product quality between two monitor wafer measurements, most manufacturers also measure product wafers, which may be sampled at strategic intervals to ensure wafer quality is maintained. However, the quality of other processed wafers beyond the measuring time is unknown. Thus, equipment abnormality may not be discovered in time, and many defective product wafers may have been produced before the next measurement, thereby resulting in a great wafer yield loss [2]. The virtual metrology (VM), which predicts the process quality of every wafer using process parameters data of production equipment without physically conducting quality measurement, can effectively overcome this problem [2]. The conjecture capability of VM allows the quality of a wafer being processed to be known instantly, and the objective of real-time wafer-to-wafer quality monitoring is possibly achieved. As such, equipment or process abnormalities can be detected promptly. Also, VM ensures adequate metrology capacity without excessive capital expenditure on metrology tools.

Since VM can provide users with total metrology information of all wafers in a FOUP by merely measuring a single wafer in the same FOUP [3] and can turn sampling inspection with metrology delay into real-time and on-line total inspection [4], VM has now become one of important topics at Advanced Equipment Control/Advanced Process Control symposiums of SEMATECH [5][6][7]. In addition, International SEMATECH Manufacturing Initiative (ISMI) has added VM into its next generation semiconductor factory realization roadmap [8]. Also, VM has been designated by International Technology Roadmap for Semiconductors (ITRS) as one of the focus areas for Factory Information and Control Systems (FICS) and advanced process control (APC) [1].

Much of VM-related literature has been published and mainly focuses on creating VM models for manufacturing processes using different algorithms or methods and illustrating the defect-detection capability or the VM conjecture accuracy. For instance, Imai and Kitabata [9] developed a VM mathematical model to perform fault detection and classification for the bath plating degradation to prevent copper interconnection failure in System on Chip (SOC). Kim and Park [10] used the radial basis function network (RBFN) to construct the VM model of a plasma etching process. Han et al. [11] utilized RBFN and genetic algorithm (GA) to model the plasma etching process. Hung et al. [2] proposed the RBFN-based VM model for the chemical vapor deposition (CVD) process. Zeng and Spanos [12] used various statistical techniques to address the challenges of modeling the plasma etching process. Su et al. [13] applied the back-propagation neural network and weighted moving average algorithm to develop VM models for plasma sputtering processes in TFT-LCD manufacturing. Gill et al. [6] adopted Kalman Filtering to conduct VM for semiconductor processes. Pan et al. [14] proposed a VM system for predicting end-of-line electrical properties of wafer in semiconductor manufacturing processes using a MANCOVA model with tools clustering. Moyne and Schulze [15] leveraged virtual metrology technology to predict yield excursions and excursion sources and employed yield prediction information as feedback to all levels of control in the fab so that processes can be continuously tuned to meet yield and device performance targets. Khan et al. [16] and Cheng et al. [4] described the VM approaches to wafer-to-wafer (W2W) control on factory level.

To facilitate automatic and fab-wide VM deployment, Cheng et al. [17] proposed the concept of automatic virtual metrology (AVM) and developed an AVM system. According to VM deployment and automation degree, the VM systems can be classified into four levels [17]:

Copyright (c) 2011 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.
Level 0: A VM system in level 0 is capable of executing off-line collection and analysis of the historical process and metrology data. It can also perform data preprocessing and create the VM conjecture model with various performance indexes, such as process data quality index (DQIX) [18], metrology data quality index (DQI) [18], reliance index (RI) [19], and global similarity index (GSI) [19].

Level 1: A VM system in level 1 should possess capabilities of on-line data collection, on-line learning, and real-time conjecturing.

Level 2: The Level-2 generic VM (GVM) system [20] aims at module pluggability. The GVM system possesses not only the on-line conjecturing capability, but also pluggable capability which enables easy replacement of its data collection driver, the VM model, and the communication agent on demand.

Level 3: The AVM system is the highest level (Level 3) VM system in terms of VM deployment and automation. An AVM system is equipped with the capabilities of automatic data quality evaluation, automatic model fanning out, and automatic model refreshing [17]. Physical characteristics of different tools are not the same. For maintaining VM conjecture accuracy, DQIX/DQIy and VM & RI/GSI models must be created based on the historical process and metrology data acquired from each chamber (a tool is usually composed of 1 to 6 chambers). Therefore, when considering fab-wide VM implementation, it is noted that the number of DQIX/DQIy and VM & RI/GSI models will increase rapidly with the growing number of tools. Under such condition, if we still take the traditional method to create those models one by one with a lot of historical data, the required huge labor cost and model-creation time will make factory-wide VM implementation either difficult or infeasible. To solve this problem, the capabilities of automatic fanning out and automatic model refreshing must be implemented in the AVM system to enable those models to be automatically spread and refreshed to the same type of tool for maintaining the VM conjecture accuracy and saving tremendous labor expenses and model-creation time.

The AVM system is very complicated. Thus, it is necessary to develop a systematic methodology for easily and systematically constructing the AVM system. However, most VM systems in the existing VM-related literatures belong to Level 1 or Level 2, which do not address the factory-wide VM deployment issue. The dual-phase VM scheme and algorithm and the data quality indexes (DQIX and DQIy), which are the underlying VM theoretical foundation of the AVM system, were developed in the literatures [21] and [18], respectively. The literature [17] described the development of an AVM system, including the procedure of creating the first set of VM models and the operational scenarios of the AVM system. The above-mentioned three literatures did not mention how to implement the AVM system in a systematic and simplified manner. Although, the literature [22] developed an AVM framework for easily constructing the AVM system, it contained only limited details. Therefore, this paper aims to develop an AVM system implementation framework (AVMSIF) to fill such a gap of the existing VM-related literatures.

The proposed AVMSIF, together with the developed server-creation approach and XML-based system-operational mechanisms, in this paper can allow the highly-complicated AVM system for factory-wide deployment to be created in a systematic and easy manner. Thus, the results of this paper can be a useful reference for industrial practitioners to construct the AVM systems. According to the proposed AVMSIF, an AVM system has been successfully created and deployed in a fifth generation thin-film-transistor–liquid-crystal-display (TFT-LCD) factory in Chi Mei Optoelectronics (CMO), Taiwan.

The rest of the paper is organized as follows. Section II introduces the used VM theoretical foundation of the AVM system: the dual-phase VM scheme and algorithm. Section III illustrates the architecture and the operational scenarios of an AVM system. Section IV presents the design of the proposed AVM system implementation framework. Section V describes the design of the control kernel. Section VI demonstrates the paradigm of AVMS implementation and testing results. Finally, Section VII is the summary and conclusions of this paper.

II. DUAL-PHASE VIRTUAL METROLOGY SCHEME AND ALGORITHM

The theoretical foundation of the proposed AVM system is based on the dual-phase VM scheme and the dual-phase VM algorithm [17][18][21], which are briefly introduced below.

A. Dual-Phase VM Scheme

The underlying dual-phase VM scheme that was developed in [17][18][21] is shown in Fig. 1. The VM server contains several functional blocks, including data preprocessing functions for process data and metrology data, a conjecture model with the dual-phase VM algorithm [17][18][21], a RI (Reliance Index) [19] module, and a GSI (Global Similar Index) [19] module.

Fig. 1. The dual-phase VM scheme [17][18][21].

The purpose of the process data preprocessing function is to compute the data quality index and the standardized values of the process data in an on-line, real-time, and automatic manner. Upon receiving an input set of process data, the process data preprocessing function uses the DQIX algorithm [18] to compute the data quality index of the process data. If an abnormality is detected, a warning signal will be sent to the process engineer for analysis and confirmation. Normal process data verified by DQIX are then standardized by Z-score and fed into the conjecture model for computing the VM results.

The metrology data preprocessing function aims to compute the data quality index and the standardized values of the metrology data for re-training or turning the conjecture model.
Whenever obtaining an actual metrology datum of a workpiece, the metrology data preprocessing function uses the DQIx algorithm [18] to compute the data quality index of that metrology datum so as to verify that both the metrology datum and its corresponding process data are normal before that metrology datum can be considered for tuning or re-training the conjecture model. First, the normality of the corresponding process data is checked by recalling their DQIy values. Then, the Z-score values of the corresponding normal process data are fed into the DQIy algorithm for detecting metrology abnormality, which may result from a measuring error or non-process factors, such as particle pollution. If an abnormality is detected, a warning signal will be sent to process engineers for analysis and confirmation. The normal metrology datum verified by DQI is then standardized by Z-score and is subsequently sent to the conjecture model for the usage of re-training or tuning the conjecture model.

**B. Dual-Phase VM Algorithm**

Fig. 2 shows the dual-phase VM algorithm [17][21] used in the VM server. Phase I emphasizes the promptness of computing VM results, whereas Phase II improves the accuracy of the VM results. The Phase-I algorithm starts to collect the process data of each processed workpiece after the conjecture model is built. The DQIy algorithm will be applied to evaluate the quality of the collected process data once process data collection of the processed workpiece is completed. If an abnormality is detected, a warning signal will be sent to the process engineer for analysis and confirmation to verify whether it is an abnormality or a clean outlier. If it is an abnormality, these process data should be discarded. The normal process data verified by DQIy are then standardized by Z-score and fed into the conjecture model to compute the VM results ($\text{VM}_j$) of the workpiece and their corresponding RI and GSI values. This computation takes about a second only; therefore, promptness is assured. By using the RI and GSI values for gauging the reliance level of the $\text{VM}_j$, the appropriateness for utilizing this $\text{VM}_j$, such as for W2W control [4], can be checked.

The Phase-II algorithm starts to collect the metrology data of the preselected workpiece in a cassette after the conjecture model is built. The correlation between the metrology data and the process data is checked via the workpiece ID once a complete set of metrology data is collected. If the correlation check is successful, the set of process data and metrology data with the same workpiece ID will be checked by the DQIy algorithm. The DQIy algorithm is used to do on-line and real-time evaluation for ensuring that both the metrology datum and its corresponding process data are normal before this metrology datum can be considered for tuning or re-training the VM models. If an abnormality is detected, a warning signal will be sent to process engineers for analysis and confirmation. If abnormality is confirmed, the metrology data will be deleted to avoid deteriorating the VM conjecture models. The normal metrology datum verified by DQI is then sent to the conjecture model for re-training or tuning usage.

Model refreshing (re-training or tuning) is essential if the first set of DQI/DQI, and VM & RI/GSI models is not generated from its own historical process and metrology data. The DQI/DQI, and VM & RI/GSI models will be updated once they are re-trained or tuned. Subsequently, the VM results ($\text{VM}_j$) and their accompanying RI/GSI values of each workpiece in the entire cassette are recomputed. After updating the DQI/DQI, and VM & RI/GSI models in Phase II, these updated models should also be adopted to compute the subsequent DQI/DQI, $\text{VM}_j$, and their accompanying RI/GSI values. Finally, the conditions for a successful and complete refreshing procedure are checked. If all of the conditions are met, which means the conjecture accuracy of the VM server is assured, the refreshing request will be disabled, and the system will enter the normal operational state. It is noted that the $\text{VM}_j$ and $\text{VM}_u$ can be applied to support W2W control [4] for the purpose of on-line advanced process control.

![Fig. 2. The dual-phase VM algorithm [17][21].](image)

**III. ARCHITECTURE AND OPERATIONAL SCENARIOS OF THE PROPOSED AVM SYSTEM**

The architecture of the proposed AVM system is shown in Fig. 3. Each set of equipment is equipped with a VM server that contains a VM conjecture model for conjecturing the process quality or faults of the equipment on-line. Also, the VM server possesses a mechanism to automatically tune or re-train the VM conjecture model so that the VM accuracy of each VM server can be maintained, and the VM server is then ready to serve various VM applications. In addition, the VM server is able to compute the quality of process data and metrology data to select good-quality data for on-line VM conjecture or refreshing the VM model.
When factory-wide VM deployment is considered, a VM manager is needed for managing all the VM servers. The VM manager shall also be designed to supervise the processes of new-model creations, fanning-out, and refreshing. The MC server is responsible for generating the first set of data quality evaluation models (DQIX/DQIy), VM conjecture models, and VM linkage evaluation models (RI/GSI) of a certain tool type. The VM manager can fan out the first set of models generated to all the VM servers of the same tool type and can monitor the real-time statuses of the VM servers. The VM client is in charge of defining configuration of VM servers, assisting the MC server in model creation, on-line monitoring the VM results, and searching the historical VM results.

Each of the AVM system components is communicating via the SEMI Interface-A like protocol [23], which enables the information among components to be shared systematically and safely. Unlike the Interface-A protocol, peer-to-peer communication is allowed among all of the servers in this AVM system.

A real example for deploying the AVM system into a fifth generation TFT-LCD factory is illustrated as follows. There are totally 18 CVD tools of the same type and each tool contains six (6) chambers. To achieve the goal of full automation, the AVM system operates as follows to automatically deploy and refresh model at runtime.

After receiving the first set of models, each VM server will initiate its advanced dual-phase VM algorithm to refresh the models in each chamber. After the refreshing process is accomplished, the VM server will report its readiness to the VM client.

The VM client prepares the data collection plan (DCP) for defining the desired VM applications, downloads the DCP to the VM server, and then activates the VM server to commence the VM applications. Finally, the VM server uploads the VM results to the VM manager according the DCP in real time. The VM manager then displays the real-time VM results on the VM client and/or stores those results into the central database for various VM applications, such as workpiece-to-workpiece quality monitoring and/or supporting W2W control, and so on.

IV. DESIGN OF AVM SYSTEM IMPLEMENTATION FRAMEWORK

According to the operational scenarios in Section III, the proposed implementation framework of the AVM system, called AVMSIF, is designed as shown in Fig. 4. The AVMSIF contains five major parts: (A) VM (Virtual Metrology) Servers, (B) Model Creation Server, (C) VM Manager, (D) Central DB (Database), and (E) VM Clients. The functions of each part are depicted in the following.

A. VM Servers

The VM server contains the following functional modules: Pluggable Data Collection, Dual-Phase Algorithm, VM Conjecture Model, VM Result Upload, VM Status Control, and OCAP & Exception Handling. These functional modules are described in the following.

(1) Pluggable Data Collection module:

The Pluggable Data Collection (PDC) module is for collecting process data and metrology data. The collected process and metrology data are then stored into the data container in the control kernel and the local database to be used in the VM conjecture model module for computing desired VM results or to be inputted to the Dual-Phase Algorithm module for tuning or retraining the VM model. The designs of the pluggable interface, the control kernel, and the data container will be depicted in Section V.
(2) Dual-Phase Algorithm module:

The dual-phase algorithm and associated indices of data quality and VM reliance have been described in Section II and are implemented in the Dual-Phase Algorithm functional module.

(3) VM Conjecture Model module:

The conjecture model is the kernel of the VM scheme. This work adopts back propagation neural network (BPNN) and multiple regressions (MR) as the algorithms for establishing the VM conjecture and reference models.

(4) VM Result Upload module:

The Upload VM Result component is used to write the prediction results of the Conjecture Model component into XML files, which are then uploaded to the VM manager.

(5) VM Status Control module:

The VM Status Control component is responsible for updating the current status of the VM server into the VM manager periodically. As such, the VM clients can monitor the status of the VM server.

(6) OCAP & Exception Handling module:

The OCAP & Exception Handling component is in charge of handling the OCAP and exceptions of the conjecture model and sending related warning messages to the VM manager.

B. Model Creation Server

The model creation server (MCS) is responsible for creating the first VM model for each type of process equipment. The MCS first acquires the data of the process and metrology equipment using its data collection interface. Next, the acquired data are filtered and preprocessed and used to create the first VM model using various built-in algorithms. The created first VM model can then be transmitted to the VM manager through the communication agent for saving into the central database and being ready to be downloaded to the assigned VM server later.

C. VM Manager

The VM manager is the operational core of the AVM system. It should possess the capabilities of system utility, OCAP management, VM management, user management, VM results processing, database management, model deployment control, alarm reporting and management, client access, and security control. The external communication of the VM server is by the communication agent.

The functional blocks of the VM manager is shown in Fig. 4. In addition to the control kernel, the VM manager contains the following components: DB Management, VM Result Processing, User Management, VM Management, OCAP (Out of Control Action Plan) Management, and System Utility. These components are described in the following.

(1) DB Management:

The DB management component possesses the functions for managing the central database, such as deleting model, getting model name, getting model information, uploading model, and uploading data collection plan (DCP), etc.

(2) OCAP Management:

The OCAP Management component is to provide OCAP related functions. It can also send messages to other systems by the alarm management system.

(3) VM Result Processing:

The VM Result Processing component is in charge of processing the prediction results and related information uploaded from the VM server. The functions of this functional...
module include parsing the VM results, inserting the VM results, OCAP handling, and database query. There is a data container, detailed later, in this component to store the VM results for supporting multi-threading functionality. Since all of the VM servers will send the VM results to this functional module for further processing, this functional module is the largest bottleneck of the VM manager.

(4) User Management:

The User Management component contains functions for managing users, such as login for users’ authentication and authorization, getting user list, adding, deleting, getting, and updating users’ information.

(5) VM Management:

The VM Management component takes care of activating and deactivating the VM servers. Once receiving a command from the VM client, it extracts the message from the command and attaches the StrategyID. Then, the command is sent to the corresponding VM server. Besides, this component can download data to the VM servers, such as AVM configurations, DCPs, models, and sensor lists.

(6) System Utility:

The System Utility component possesses functions for setting the system, such as AVM registration, AVM refreshing, AVM monitoring, OCAP messaging, Saving EP, uploading dispatch information, uploading sensor definition, and downloading reply.

D. Central DB

The central database is used to centrally store various types of data, including the first VM models created by the MCS and the VM results from the VM servers.

E. VM Clients

The VM clients provide various user-friendly graphical user interfaces (GUI) for the users to operate the AVM system. The GUIs of the VM clients can be utilized to perform functions of status monitoring and inquiry, data collection configuration, AVM configuration, and MCS configuration, and so on. The VM clients use the communication agent to communicate with the VM manager.

V. DESIGN OF THE CONTROL KERNEL

In order to let the functional components of the VM Manager, VM Server, and MC Server be modular and operate in a plug-and-play manner, we propose a “one control kernel plus add-on functions” approach to design the VM Manager, VM Server, and MC Server. In other words, we first design the Control Kernel that contains common core functions of these three types of Servers and generic interfaces that can enable functional components to be plug-and-play. Then, the VM Manager, the VM Server, and the MC Server can be easily constructed by integrating the Control Kernel with their respective add-on functional components through the plug-and-play interfaces, as shown in Fig. 5. Fig. 5 (a) and 5 (b) illustrate the Control Kernel and functional components of VM Server and VM Manager, respectively. Thus, the Control Kernel plays a key role in each Server. The functional blocks of the Control Kernel is shown in the lattice area of Fig. 5. The Control Kernel contains plug-and-play interfaces and four common core functional components of the Servers: Communication Agent, Data Container, Server Initialization, and Time Synchronization. In the following, we first describe the design of plug-and-play interfaces and the scenario rules used in the Control Kernel. Then, the design of the four common core functional components is depicted.

![Diagram of Control Kernel and functional components](image-url)
explained as follows: The lock() and unlock() are used to lock
and unlock the thread, respectively, to ensure that only one
object to access the Data Container at a time. The setPlanRoot()
and getPlanRoot() are used to set and get the location of
the root tag of the Data Container, respectively. The getSize() is
used to get the size of the Data Container. The getValue() and
setValue() are used to get and set values from and to the Data
Container, respectively. The msz_DataClassName attribute is
used to store the name of the executing ActionDll. The
DataContainer class inherits the CBaseDataClass and is used to
declare the Data Container object in ActionDlls. The
st_systemIni attribute is a structure that stores the information
of the system setting file system.ini, introduced later.

The second argument is used to accept the returned data of
executing the ActionDll, and its type is DWORD. The third one
is the CCommandMessage class which can be used to store
XML-based commands, illustrated later, to be performed by the
ActionDll. Each ActionDll uses the CCommandMessage class
to create the command message object. The m_Primary
attribute is used to store the content of the received XML
command message in the form of a string. The ActionDll can
use getPrimary() to get the associated command to be executed.

Thus, by inheriting the BaseAction class and defining the
content of the Execute() method, each ActionDll will possess
the generic Execute() method that has the same arguments, but
has different contents for achieving the functionality of each
ActionDll. Then, for performing an ActionDll, the Control
Kernel can just call the Execute() method of that ActionDll,
thereby allowing ActionDlls to be plug-and-play.

"command" element is used to store the command name. The
"parameters" element is used to host the information of
parameters used by the command. For instance, in Fig. 7, the
message name is AVMRegistration, the schema is located in
"\XSC\Format\AVMRegister.xsd," the command name is
"SystemUtilityProcess," and the information of parameters
used by the command is local information. Since every
command has its own required parameters, the "parameters"
attribute can contain many "parameter" sub-elements. For
instance, some of parameters and their values used by the
"SystemUtilityProcess" command in Fig. 7 are described as
follows: The "AVMName" parameter contains the name of the
VM Server, and its value is "AVM_81." The "IsActivate" parameter
contains the status of the VM Server, and its value is
"Deactivate." The "Chamber" parameter contains the ID of the
chamber monitored by the VM Server, and its value is "123." The
"Process" parameter and the "ProductID" parameter contain
the information of the process type and the Product ID
of the equipment monitored by the VM Server. The "HostIP"
parameter contains the IP address of the VM Server, and its
value is "140.116.86.164." The "DCP" parameter contains the
status whether the VM Server has data collection plans or not,
and its value is "No." The "InitialModel" parameter contains
the status whether the VM Server has an initial VM model or
not, and its value is "NULL." Finally, the "Configuration"
parameter contains the status whether the VM Server has a
system configuration file or not, and its value is "No." Other
command messages can be designed in a similar way.

B. Design of Scenario Rules Mechanism

In the proposed AVM system, the command messages are
designed in the form of XML [24] files. The design concept of
the XML command messages is depicted below.

Fig. 7 shows a sample XML command message that is sent to
the VM Manager from a VM Server for registering its
information. Each XML command message contains three key
elements: "message," "command," and "parameters." The
"message" element is the root of the XML message and
contains the information of the schema location, the message
name, the source where the command comes from, etc.
To complete a task, a Server in the AVM system may require performing several functional components (ActionDlls) in a specific order. One way to complete such a task is to call respective ActionDlls in the required order by hardcoding in the source code of the program. However, by this approach, if the system needs to add a new task, we must revise the source code and rebuild the program, which is cumbersome and time-consuming.

In this study, we develop a "scenario rules" mechanism to solve the above problem. For completing each task, a corresponding scenario rule is added in the scenario XML file, Scenario.xml. A sample scenario XML file is shown in Fig. 8. For example, to complete the task of registering a VM Server, whose corresponding XML command is shown in Fig. 7, a scenario rule is added in the Scenario.xml, as labeled by the arrow lines in Fig. 8. After receiving the XML command message of Fig. 7, the Control Kernel retrieves the command name "SystemUtilityProcesss" from the message and then searches the rule with the value equaling "SystemUtilityProcesss" in the Scenario.xml of Fig. 8. Next, according to the matched rule, the functional component "SystemUtilityActionDll," contained in the "Action" element, is executed. The attribute "value" of the "Deep" element is used to host the step number. Since this is a one-step rule, the "value=1" is set in the "Deep" element.

A more complicated sample rule that is used to complete the task of Phase-I VM Conjecture is shown in Fig. 9. For the sake of easy explanation, the flowchart of Phase-I VM Conjecture is also shown on the left hand side of the figure. There five steps in the sample rule. In Step 1 (R1), the process data collection of a certain workpiece is performed, This action refers to the content of the first "Deep," which executes the DataCollectionActionDll, the functional component for collecting data. In Step 2 (R2), there are two <Deep value="2"> elements with different values of the "Preformation" element. The attribute "value" of the "Preformation" element is used to compare with the returned value of executing the ActionDll in the previous step. In Fig. 9, if the returned value of executing the DataCollectionActionDll is 1, meaning that the data collection is not finished yet, then the action "JUMP" contained in <Preformation value="1"> is executed to continue the data collection; whereas if the returned value is 0, meaning that the data collection is completed, the Control Kernel performs the DQIxAlgorkthmActionDll, the functional component for computing the Data Quality Index of process data. Similarly, in Steps 3 (R3), the GSIActionDll is executed for computing the Global Similarity Index. In Step 4 (R4), the ConjectureActionDll is executed for computing the Reliance Index and the Phase-I VM results.

It is noted that if the value of the <Action> element is "JUMP," the <Criterior> elements needs to be used. The first <Criterior> element is used to include the name of the command that the Control Kernel should jump to execute. If the value of the <Action> element is not "JUMP," the "Action1" value is set to the first <Criterior>, meaning no any effects. The value of the second <Criterior> element is used to indicate the waiting time (unit: second) before executing the ActionDll. For example, in Step 5 (R5) of Fig. 9, the Control Kernel should jump to execute the "UploadVMresult" command immediately, i.e. no waiting time.

Thus, by using the developed "scenario rules" mechanism, we can integrate various ActionDlls in certain orders or flows to complete various tasks by just editing the rules in the Scenario.xml file. Then, the Control Kernel can follow the rules to execute the commanded tasks in run time without needing to rebuild the system, thereby effectively overcoming the afore-mentioned shortcomings of the traditional hardcoding approach.

C. Design of Data Container Mechanism

Each server of the AVM system contains several ActionDlls. To complete a specific task, these ActionDlls must collaborate with each other. Hence, there exists the problem of accessing common data across ActionDlls. For instance, in a VM Server, the Data Pre-process component computes the data quality of process data and sends standardized good-quality process data to the VM Conjecture Model component for getting VM results and to the Reliance Index component for evaluating the reliance of the VM results. Thus, the programs of these related components, called related programs hereafter, need to access some common data.

To tackle this problem, traditionally we can declare a global structure variable in the program of the Control Kernel to allow the plugged ActionDlls to store and retrieve common data. However, such a hardcoding approach has the following shortcomings: Firstly, whenever a new global structure variable is added in the program of the Control Kernel, we need to rebuild the Control Kernel. Secondly, declaring too many global variables is a risky programming practice, which may cause that changing the value of a global variable incursively results in erroneous results of some methods. Thirdly, each ActionDll can be developed independently by different persons. To create a new component that will use the common data of the existing components, the developer needs to refer to the source codes of the Control Kernel for obtaining the formats of the common data. If the source codes are unavailable, this approach fails. Even though the source codes are available, source code sharing is unsafe for the system and is usually prohibited.
Thus, to solve the above-mentioned problem of accessing common data across ActionDills, we develop a “Data Container” mechanism in this study. Fig. 10 shows a sample schema XML file of a Data Container. In the AVM system, a Data Container is used to store data carried by command messages for being used by the ActionDill. During the system initialization phase, the Control Kernel will load the system.ini file, described later, that contains the pathname of the schema XML file of the Data Container. Then, according to the loaded schemas, the data structure of the Data Container object can be automatically created for storing and getting the data of command messages. For example, the schema XML file of Fig. 10 has five-layer tags. The tags of Layers 1 to 5 are “equipment,” “module,” “unit,” “parameters,” and “parameter,” respectively. Also, the names of “equipment” and “module” are “AVM1” and “ConjectureInformation,” respectively. There are two names for the third-layer tag “module”: “ConjectureModelData” and “Information.” There several names for the fourth-layer tag “parameters” and the five-layer tag “parameter.” Therefore, by loading the schema XML file of Fig. 10, the Control Kernel or each ActionDill can automatically create a Data Container object that is an array of 5-dimensional Vectors. The data access of the Data Container object will look like as follows:

(1) Getting the first value of the “StrategyID” dimension of the Data Container:
\[
\text{a = } [\text{AVM1}][\text{ConjectureInformation}][\text{ConjectureModelData}][\text{RISum}][\text{StrategyID}].\text{Value}[0];
\]

(2) Storing 0.5 to the second value of the “Threshold” dimension of the Data Container:
\[
\text{a = } [\text{AVM1}][\text{ConjectureInformation}][\text{ConjectureModelData}][\text{RISum}][\text{Threshold}].\text{Value}[1] = 0.5;
\]

By using the developed “Data Container” mechanisim, the Server in the AVM system can create a new Data Container or update the schema of an existing Data Container by just restarting the Server and loading the new or updated XML schema file of the Data Container, without needing to rebuild the Control Kernel. Thus, the developer can follow this approach to flexibly load various Data-Container XML schema files to create desired Data Containers to fulfill the requirements of accessing common data across different ActionDills.

D. Design of Time Synchronization

The Time Synchronization component is used to synchronize the time of the whole AVM system. The VM Manager will be executed first. Then, other Servers will adjust their time to be the same as that of the VM Manager.

E. Design of Communication Agent

The Communication Agent component is responsible for sending/receiving commands and data to/from outside systems. The communication interface of the Communication Agent is designed to conform to the SEMI Equipment Data Acquisition (EDA) Standards [23], also known as Interface A. The major SEMI EDA standards include E120 (Common Equipment Model), E125 (Equipment Self-Description), E132 (Equipment Client Authentication and Authorization), and E134 (DCM, Data Collection Management). Based on the XML technology...
Interface A allows secure and flexible communications among VM Clients, MC Server, VM Manager, and VM Servers through SOAP (Simple Object Access Protocol) [25].

Fig. 10. A sample schema XML file of a Data Container.

**F. Design of Server Initialization**

The Server Initialization component can load the associated system initialization setup information into the system, which is usually stored in the file `system.ini`. Fig. 11 shows a sample `system.ini` file, which is a text file. The `system.ini` file includes the information of system setting ([Setting]), database connection setting ([DB]), communication ports setting ([InterfaceA]), scenario setting ([Scenario]), common equipment model (CEM) setting ([CEM]), command message setting ([Command]), and so on.

![Fig. 11. A sample System.ini file.](image)

**VI. PARADIGM AVMS IMPLEMENTATION AND TESTING RESULTS**

According to the proposed AVMSIF, a paradigm AVM system has been successfully created and deployed in a fifth generation thin-film-transistor–liquid-crystal-display (TFT-LCD) factory of Chi Mei Optoelectronics (CMO) in Taiwan. The following describes the software and hardware used in the implementation, the scenarios of the integrated tests, and the testing results.

**A. Implementation of the Paradigm AVM Systems**

The hardware and software for developing the paradigm AVM systems are described below.

(1) Software:

- Operating system: Windows XP (SP2).
- Programming language: C++.
- Database: Oracle 10g for constructing Central DB.

(2) Hardware:

Four desktop PCs with Intel Pentium 4 CPU are used to implement VM Manager, VM Client, VM Server, MC Server, MC Client and Central DB.

**B. Scenarios for Integrated Tests**

In order to verify the effectiveness of the proposed AVMSIF, various testing scenarios are designed to test the paradigm AVM systems as described below:

(1) Activating a VM server:

The testing steps for activating a VM server are as follows:

**Step 1:** The VM client selects a desiring VM server and sends the activation command to the VM manager.

**Step 2:** The VM manager searches the information of the selected VM server.

**Step 3:** The VM manager activates the selected VM server.

**Step 4:** The VM manager sends the message that describes the activation succeeds or fails to the VM client.

(2) Deactivating a VM server:

The testing steps for deactivating a VM server are as follows:

**Step 1:** The VM client selects a desiring VM server and sends the deactivation command to the VM manager.

**Step 2:** The VM manager searches the information of the selected VM server.

**Step 3:** The VM manager deactivates the selected VM server.

**Step 4:** The VM manager sends the message that describes the deactivation succeeds or fails to the VM client.

(3) Fanning out (Deploying) a conjecture model:

The testing steps for fanning out a conjecture model to selected VM servers are as follows:

**Step 1:** The VM client selects a conjecture model and target VM servers and sends the deployment command to the VM manager.

**Step 2:** The VM manager searches the information of the selected VM servers.

**Step 3:** The VM manager activates the selected VM server.

**Step 4:** The VM manager sends the message that describes the activation succeeds or fails to the VM client.
conjecture model from the Central DB.

Step 3: The VM manager downloads the conjecture model to the selected VM servers.

Step 4: The VM manager sends the message that describes the deployment succeeds or fails to the VM client.

(4) Downloading a DCP (Data Collection Plan) to a VM server:

The testing steps for downloading a DCP to a selected VM server are as follows:

Step 1: The VM client selects a DCP file and a desiring VM server and sends the downloading command to the VM manager.

Step 2: The VM manager downloads the DCP file to the selected VM server.

Step 3: The VM manager sends the message that describes the downloading succeeds or fails to the VM client.

(5) Uploading VM results:

The testing steps for uploading the conjecture results of a VM server to a VM client are as follows:

Step 1: The VM server sends the uploading command with the conjecture results to the VM manager.

Step 2: The VM manager parses and classifies the conjecture results and extracts the information of the selected VM client from the command.

Step 3: The VM manager uploads the conjecture results to the selected VM client.

Step 4: The VM manager stores the conjecture results into the Central DB.

C. Integrated Testing Results

The GUIs and the testing results of the paradigm AVM system are described as follows.

(1) GUIs of the AVM system:

The major GUIs of the paradigm AVM system are constructed on the VM Client side. For the sake of space limit, just The GUIs for activating VM Servers and uploading VM Results are displayed below.

1) GUI for Activating VM Server: The GUI for activating VM servers is shown in Fig. 12. In the equipment list, the deactivated AVMs are labeled with switched-off bulbs. Once an AVM is doubly clicked, it will be added in the activated list at the bottom of the right hand side for waiting to be activated. After the user confirms the activated list and presses the “Submit” button in the middle, the AVMs in the activated list will be activated. The corresponding acknowledgement message from the VM manager will be recorded in the system log and displayed on the log GUI.

2) GUI for Uploading VM Results: When the button in the main GUI is pressed, the GUI for uploading VM results is shown as in Fig. 13. The user can monitor the real-time prediction results through this GUI. After the desired VM server and chamber are selected, the VM Client will send the uploading command to the VM manager. Then, the corresponding VM results will be sent the VM Client from the VM manager and instantly displayed in the GUI in Fig. 13. When the user clicks on the data at the bottom of the left hand side, the corresponding ISI information will be plotted and displayed at the bottom of the right hand side.

(2) Testing Results:

The testing results of the AVM system are shown and evaluated in the following four aspects: deployment time saving, improvement of factory operations, automatic VM model refreshing, and defect detection.

![GUI for activating VM servers.](image-url)
Deployment Time Saving:
The deployment of a VM system contains three stages: VM environment setting, model creation, and online VM conjecture with model refreshing. Since the main advantage of the AVM system with its implementation strategy is that it minimizes the manual setting up of the traditional separate VM schemes, the deployment time saving of the AVM system compared to the traditional VM system is analyzed as follows.

The deployment time, $T_{\text{VM Deployment}}$, of the traditional VM system can approximately computed as follows:

$$T_{\text{VM Deployment}} = n_{\text{VM Server}} \times t_{\text{setting}} + n_{\text{Chamber}} \times t_{\text{Modeling(VM)}}$$  \hspace{1cm} (1)

where $n_{\text{VM Server}}$ is the number of the VM servers, $t_{\text{setting}}$ is the time spent in setting a VM Server, $n_{\text{Chamber}}$ is the number of chambers needed to be modeled, and $t_{\text{Modeling(VM)}}$ is the time taken to create a set of models (including conjecture models, reliance index, etc.) for a chamber using the traditional VM system. Recall that a VM server can connect and monitor a piece of equipment that usually contains several chambers.

On the other hand, the deployment time, $T_{\text{AVM Deployment}}$, of the AVM system can be roughly expressed as follows:

$$T_{\text{AVM Deployment}} = n_{\text{VM Server}} \times t_{\text{setting}} + t_{\text{Modeling(AVM)}} + n_{\text{Refresh Count}} \times \Delta t_{\text{Sampling}}$$  \hspace{1cm} (2)

where $t_{\text{Modeling(AVM)}}$ is the time required to create the first set of models for a chamber using the AVM system, $n_{\text{Refresh Count}}$ is the number of metrology data samples for the automatic VM model refreshing scheme of the AVM system to get a conjecture model with satisfied accuracy, and $\Delta t_{\text{Sampling}}$ is the time between two workpiece measurements. Although the setting time of a VM server in the traditional VM system and the AVM system may be different, here we assume they are the same.

The deployment time saving, $T_{\text{Saving}}$, of the AVM system compared to the traditional VM system is then computed as follows:

$$T_{\text{Saving}} = T_{\text{VM Deployment}} - T_{\text{AVM Deployment}}$$

$$= n_{\text{Chamber}} \times t_{\text{Modeling(VM)}} - t_{\text{Modeling(AVM)}} - n_{\text{Refresh Count}} \times \Delta t_{\text{Sampling}}$$  \hspace{1cm} (3)

In the following, the real example for deploying the AVM system into a fifth generation TFT-LCD factory mentioned in Section III is used to illustrate the deployment time saving of the AVM system. There are totally 18 CVD tools of the same type and each tool contains 6 chambers. Thus, $n_{\text{VM Server}} = 18$ and $n_{\text{Chamber}} = 18 \times 6 = 108$. The value of $t_{\text{setting}}$ depends on the complexity of the VM server and the user’s familiarity with operating the system’s GUIs. In our experience, $t_{\text{setting}}$ usually ranges from 3 to 10 minutes. Here, $t_{\text{setting}} = 0.1$ hr is used in the computation. By adopting the traditional VM system, we need to create a set of VM models for each chamber individually. By contrast, the AVM system can automatically deploy and refresh the first set of DQI/DQII and VM & RI/GSI models to all of the 108 chambers of the CVD tools.

The model creation includes three sequential operations: data collection followed by data pre-processing followed by creating and testing model. The value of $t_{\text{Modeling(VM)}}$ depends on the number of metrology parameters, the number of process parameters, the number of modeling data sets, and the number of testing data sets. In our own experience, for a case that involves 15 metrology parameters, 25 process parameters, 500 modeling data sets, and 100 testing data sets, $t_{\text{Modeling(VM)}}$ is about 10 hr. Both of the data-collection operation and the creating-and-testing-model operation can completed in less than 10 minutes, individually. It is the data pre-processing operation that dominates $t_{\text{Modeling(VM)}}$ the most. This is because the data pre-processing operation involves many manual tasks, such as screening out all the process and metrology data produced during periodic maintenance or product shift, establishing process data standard patterns, selecting proper...
indicators, constructing metrology data abnormal modes, evaluating the data quality of process indicator data and metrology data and deleting the abnormal ones. Thus, if we apply the traditional VM system to this case, the deployment time can be approximated by Eq. (1) as follows:

\[ T_{\text{VM Deployment}} = 18 \times 0.1 + 108 \times 10 = 1081.8 \text{ hr} \approx 45 \text{ days} \quad (4) \]

Since the AVM system possesses DQIx/DQIy models in the data pre-processing operation for automatically evaluating the data quality of process indicator data and metrology data and deleting the abnormal ones, \( t_{\text{Modeling(AVM)}} \) is much less than \( t_{\text{Modeling(VM)}} \). In this case, \( t_{\text{Modeling(AVM)}} \) is about 3 hr. In general, for the CVD process, a cassette contains 30 pieces of glass, the production time of a cassette of glass is about 0.5 hr, and sampling rate is one out of 20 cassettes. Therefore, \( \Delta t_{\text{Sampling}} \approx 20 \times 0.5 = 10 \text{ hr} \). According to our experience, \( n_{\text{Refresh Count}} \) is about 3 [17]. The deployment time of the AVM system in this case can then be computed by Eq. (2) as follows:

\[ T_{\text{AVM Deployment}} = 18 \times 0.1 + 3 \times 3 + 10 = 34.8 \text{ hr} \approx 1.45 \text{ days} \quad (5) \]

In this case, the deployment time saving, \( T_{\text{Saving}} \), of the AVM system compared to the traditional VM system is about 1047 hr (about 43.6 days) by Eq. (5). Thus, the AVM system can significantly reduce the total deployment time of factory-wide VM, about 1.45 days (AVM System) vs. 45 days (Traditional VM System).

### 2) Improvement of Factory Operations:

Since the AVM system is generic, in addition to the TFT-LCD factory, the AVM system was also deployed to a Solar Cell manufacturing factory in Taiwan. The following shows a real case that the AVM system did improve the factory operation by helping the engineers timely detect the parameter abnormalities of the ARC (Anti-Reflection Coating) process and identify the key parameter causing the abnormalities. Consequently, the engineers were then able to conduct maintenance on the abnormal component to restore the PECVD (Plasma Enhanced Chemical Vapor Deposition) equipment to the normal status.

Recall that the AVM system provides RI (Reliance Index), GSI (Global Similarity Index), and ISI (Individual Similarity Index) [19]. The RI is for evaluating the reliance level of a VM conjecture result. The GSI is defined as the degree of similarity between the input of process data and the historical set of process data used for establishing the conjecture model. The ISI of an individual process parameter is defined as the degree of similarity between this individual process parameter’s standardized process data and the historical set of process data that are used for establishing the conjecture model. If the GSI of an input set of process data is greater than the GSI, it indicates that some deviations may occur in the process data owing to the high GSI. Then, the parameter(s) with the greatest deviation can be identified from the corresponding ISI Pareto chart [19].

The ARC process is a key process in Solar Cell manufacturing. The anti-reflection coating can enhance the cell's power conversion efficiency, which is an important KPI for solar cells. In general, for the ARC process, a boat contains 144 wafers, the production time of a boat of wafers is about 40 minutes, and the sampling rate is one out of 10 boats. Thus, \( \Delta t_{\text{Sampling}} = 10 \times 40 / 60 = 6.67 \text{ hr} \). The statistics of the event log of a PECVD tool in the ARC process is shown in Table 1 and described as follows.

\[ t = ET_1 \quad \text{At} \quad ET_1, \quad \text{the AVM system detected that the GSI (66) of an input set of the process data was greater than GSI (9), meaning that some deviations may occur in the process data. Further, the corresponding ISI Pareto chart indicated that the tube pressure is the key process parameter causing this high GSI. The corresponding conjecture values are normal. The AVM system then sent an alarm by e-mail to the engineer in charge.} \]

**Table 1 Statistics of the event log of a PECVD tool in the ARC process.**

<table>
<thead>
<tr>
<th>Event Time</th>
<th>Number of Processed Boats</th>
<th>Number of Processed Boats with GSI &gt; GSI</th>
<th>Number of Processed Boats with GSI &lt; GSI</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t = ET_1 )</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( ET_1 &lt; t \leq ET_2 )</td>
<td>21</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>( ET_2 &lt; t &lt; ET_3 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t = ET_3 )</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( ET_3 &lt; t \leq ET_4 )</td>
<td>6</td>
<td>0</td>
<td>6</td>
</tr>
</tbody>
</table>

**ET_1 < t < ET_2:** This period is about 4 hours, during which the engineer stopped the PECVD equipment to clean and maintain the gas pressure valve. The engineer found out that the gas pressure valve broke down, thereby confirming the effectiveness of the GSI. Then the engineer replaced the gas pressure valve.

\[ t = ET_4 \quad \text{At} \quad ET_4, \quad \text{the PECVD equipment processed a boat of wafers. The corresponding GSI values were less than the GSI, and the gas pressure values had restored to be normal.} \]

**ET_4 < t < ET_5:** This period is about 4 hours, during which 6 boats of wafers were processed. All of the corresponding GSI values were less than the GSI, indicating that the process had become stable.

### 3) Automatic VM Model Refreshing:

One of the key merits of the AVM system is the capability of automatic and fab-wide VM deployment by applying the automatic model refreshing algorithm. An illustrative example demonstrating the automatic-VM-model-refreshing capability of the AVM system for CVD tools in a CMO’s fifth-generation TFT-LCD factory can be found in Fig. 5 and Fig. 6 of [17]. The good automatic-VM-model-refreshing capability of the AVM system can save a lot of time in maintaining the model accuracy after the VM deployment.
4) Defect Detection:

The defect detection is one of key capabilities of the VM technology. An illustrative example demonstrating the defect-detection capability of the dual-phase VM scheme for CVD equipment can be found in Fig. 3 of [21]. The good defect-detection capability of the AVM system can effectively facilitate the process and equipment engineers to find and fix the potential abnormalities in process or equipment.

VII. SUMMARY AND CONCLUSIONS

Automatic virtual metrology (AVM) is the highest-level (Level 3) technology for VM applications from the perspective of automation. It is an enabler for fast applying VM to all pieces of equipment in a factory. This paper aims to present the development of an AVM system implementation framework (AVMSIF) for high-tech industries, such as Semiconductor and TFT-LCD industries. First, we introduce the concept of AVM and show the used VM theoretical foundation: the dual-phase VM scheme and algorithm. Next, the architecture and the operational scenarios of an AVM system are drawn up for being as the blueprint of the system framework design. Then, an AVM system implementation framework, called AVMSIF, is designed to fulfill the functionality and operational scenarios of the proposed AVM system. The AVMSIF consists of five parts: a model creation server (MCS), many VM servers, a VM manager, VM Clients, and a central database (Central DB), and possesses the capabilities of an AVM system, including creating VM models, deploying and refreshing VM models, on-line VM conjecturing, monitoring and managing VM servers remotely, storing model data and conjectured results centrally, and providing various friendly graphical user interfaces.

To facilitate the creation of the servers of the AVM system, we propose a “one control kernel plus add-on functions” approach in this paper. By adopting the proposed approach, the VM manager, the VM servers, and the MC server can be easily constructed by just integrating a control kernel with their respective add-on functional components through the plug-and-play interfaces of the control kernel. Besides, we use the XML technology to develop the “scenario rules” mechanism and the “data container” mechanism in the control kernel. Specifically, we define a scenario rule that specifies which functional components should be executed in which orders for completing a given task in a scenario XML file. Thus, the server with the control kernel can load the scenario XML file and follow the scenario rules to complete various the commanded tasks in run time without needing to rebuild the system. Similarly, by using the “data container” mechanism, the server with the control kernel can load XML schema files to create desired data structures that can store common data across different functional components in run time. Therefore, when any tasks or data items need to be added into the server, these two XML-based mechanisms that require only editing and loading XML files can effectively overcome the shortcomings of the traditional hardcoding approach that needs to rebuild the system. Also, by adopting plug-and-play interfaces and desired functional modules, the AVMIF can be applied to different types of equipment in the factory-wide VM deployment.

According to the proposed AVMSIF in this paper, an AVM system has been successfully created and deployed in a fifth generation thin-film-transistor–liquid-crystal-display (TFT-LCD) factory in CMO and a Solar Cell manufacturing factory in Taiwan. Testing results show that the AVM system can significantly reduce the total deployment time of factory-wide VM, improve the operation of a manufacturing plant following the implementation of the AVM scheme, save a lot of time in maintaining the model accuracy after the VM deployment, and effectively facilitate the process and equipment engineers to find and fix the potential abnormalities in process or equipment, thereby confirming the effectiveness of the proposed AVMSIF.

The AVM system is very complicated. Thus, it is necessary to develop a systematic methodology for successfully constructing the AVM system. However, the existing VM references mainly focus on creating VM models using different algorithms or methods and illustrating the defect-detection capability or the VM conjecture accuracy. Only a few of them mentioned how to implement the AVM system, but with limited details. By contrast, the proposed AVMSIF, together with the developed server-creation approach and XML-based system-operational mechanisms, in this paper can allow the complex AVM system to be created in a systematic and easy manner. Thus, the research results of this paper can fill such a gap and can be a useful reference for high-tech industries, such as Semiconductor, Solar Cell, and TFT-LCD systems, to construct their AVM systems.

REFERENCES

Control of Mobile Robots, and Microprocessor and Embedded Systems. Metrology, RFID and Wireless Sensor Networks, Dynamic Simulation and His expertise is in the fields of e-Manufacturing, e-Diagnostics, Virtual Defense Science, CCIT, NDU, between September 2009 and July 2010. Since and Electronic Engineering, CCIT, National Defense University (NDU), and was promoted to Associate Professor and Professor in 2000 and 2006, respectively.


Min-Hsiung Hung (S’96-M’99-SM’06) was born in Changhua County, Taiwan, Republic of China (R.O.C.), on September 9, 1966. He received the B.S. degree from Chung Cheng Institute of Technology (CCIT), Taoyuan, Taiwan, R.O.C., the M.S. degree from National Cheng Kung University, Tainan, Taiwan, R.O.C., and the Ph.D. degree from The Ohio State University, Columbus, Ohio, U.S.A., in 1988, 1992, and 1999, respectively, all in electrical engineering. He was with the Department of Electrical Engineering, CCIT, as a Lecturer in 1992, and was promoted to Associate Professor and Professor in 2000 and 2006, respectively. He also served as the Chairman of the Department of Electrical and Electronic Engineering, CCIT, National Defense University (NDU), Taiwan, from September 2007 to August 2009, and as the Dean of School of Defense Science, CCIT, NDU, between September 2009 and July 2010. Since August 2010, he is with the Department of Computer Science and Information Engineering, Chung Cheng University, Taiwan, Taiwan, R.O.C., as a Professor. His expertise is in the fields of e-Manufacturing, e-Diagnostics, Virtual Metrology, RFID and Wireless Sensor Networks, Dynamic Simulation and Control of Mobile Robots, and Microprocessor and Embedded Systems.

Chun-Fang Chen (S’12) was born in Hsinchu, Taiwan, Republic of China, on January 17, 1985. In 2005 and 2006, he respectively received the B.S. degree from Industrial Engineering and Management Dept., National Taipei University of Technology, Taipei, Taiwan. He is currently a candidate for doctor's degree of the Institute of Manufacturing Information and Systems, National Cheng Kung University. His current research interests include Automatic Virtual Metrology, factory automation for semiconductor manufacturing, equipment management system, e-Manufacturing, embedded systems.

Hsien-Cheng Huang (S’07) was born in Tao-Yuan, Taiwan, Republic of China, on April 25, 1979. In 2002 and 2004, he respectively received the B.S. degree from Industrial Engineering and Technology Management Dept., Da-Yeh University, Chang-Hua, Taiwan and the M.S. degree from Institute of Manufacturing Information and Systems, National Cheng Kung University, Tainan, Taiwan. In 2007, he is currently a candidate for doctor's degree of the Institute of Manufacturing Information and Systems, National Cheng Kung University. His current research interests include Automatic Virtual Metrology, factory automation for semiconductor manufacturing, equipment engineering system, e-Manufacturing, embedded systems, and short range wireless technology.

Haw-Ching Yang (M’04) was born in Tainan, Taiwan, R.O.C., in 1966. He received the B.S. degree in engineering science from National Cheng Kung University, Tainan, in 1989, the M.S. degree in control engineering from National Chiao Tung University, Hsinchu, in 1991, and the Ph.D. degree in the Institute of Manufacturing Engineering from the National Cheng Kung University, in 2003. He was with Automation Engineering at Da-yeh University, as a lecturer from 1997 to 2000. Since September 2004, he has been an Assistant Professor of Institute of System and Control at the National Kaohsiung First University Science and Technology, Kaohsiung. He is with experiences in the analysis, modeling, design, and implementation of manufacturing automation. His research interests include production prediction, system optimal, virtual production control, intelligent system, and cloud-based applications.

Fan-Tien Cheng (S’87-M’89-SM’98-F’08) received his B.S. degree from National Cheng Kung University (NCKU), Taiwan, R.O.C., in 1982 and his Master’s (1989) degrees from The Ohio State University, USA, all in Electrical Engineering.

Dr. Cheng is, currently, Chair Professor of NCKU. He served as the Director of the Institute of Manufacturing Engineering (IME), NCKU (Aug. 1998 – July 2001) He built a web-enabled experimental Manufacturing Execution System and a Supply Chain Information System for IC Packaging. He also established test beds for e-Diagnostics, Equipment-Engineering-System, engineering-Chain- Management-System, and Automatic-Virtual-Metrology frameworks for semiconductor manufacturing at IME Automation Laboratory for educational and research purposes. His research interests include Semiconductor Manufacturing Automation, e-Manufacturing, Virtual Metrology, and Intelligent Predictive Maintenance.

Prof. Cheng received the Senior Scientist Award from the DoD, ROC (1994). He won the Kayamori Best Automation Paper Award at IEEE ICRA 1999. He also received Outstanding Industry-University-Cooperation (IUC) Award from the MoE, ROC (2003); NCKU Distinguished IUC-Professor Awards (2004 & 2008); ROC National Science Council (NSC) Outstanding IUC Award (as the only awardee in 2006); ROC NSC Outstanding Research Award (2006 & 2009); University Industry Economy Contribution Award - Individual Award from the Ministry of Economic Affairs, ROC (2008); the TECO Award, TECO Technology Foundation, ROC (2010); the 2011 National (Silver) Invention and Creation Award from MOEA, ROC; and the 2011 Award for Outstanding Contributions in Science and Technology from the Executive Yuan, ROC.

Prof. Cheng served as Associate Editor of IEEE Transactions on Robotics and Automation (2000-2004) and IEEE ICRA 2006 Kayamori Best Automation Paper Award Committee Chair. He, currently, serves as the Senior Program Committee member of ICRA 2011 and the Program Chair of IEEE WCICA 2011. He was the Convener and Program Director of the NSC Automation Engineering Program, Taiwan, ROC (2007-2009). Prof. Cheng has founded the e-Manufacturing Research Center (eMRC) at NCKU since Jan. 2008 and, currently, serves as the Director of eMRC.