Cluster tool module communication based on a high-level fieldbus

JIN-HWAN LEE, TAE-EOG LEE and JEONG-HYEON PARK

Abstract. A cluster tool for semiconductor manufacturing is an integrated device that consists of several single wafer processing modules and a wafer transport module based on a robot. The distributed module controllers are integrated by an inter-module communication network and are coordinated by a centralized controller, called a cluster tool controller (CTC). Since the CTC monitors and coordinates the distributed complex module controllers for advanced process control, complex communication messaging and services between the CTC and the module controllers are required. A SEMI standard, CTMC (cluster tool module communication), specifies application-level communication service requirements for inter-module communication. We propose the use of high-level fieldbuses–for instance, PROFIBUS-FMS–for implementing CTMC since the high-level fieldbuses are well suited for complex real-time distributed manufacturing control applications. We present a way of implementing CTMC using PROFIBUS-FMS as the communication enabler. We first propose improvements of a key object of CTMC for material transfer and the part transfer protocol to meet the functional requirements of modern advanced cluster tools. We also discuss mapping objects and services of CTMC to PROFIBUS-FMS communication objects and services. Finally, we explain how to implement the mappings.

1. Introduction

A cluster tool is an integrated semiconductor manufacturing device that consists of several single-wafer processing modules (PMs), a wafer transport module (TM), and cassette modules (CMs) (see figure 1(a)). A TM consists of a robot for wafer handling and slot valves through which wafers are loaded into, or unloaded from, the PMs. A robot has a single arm or dual opposite arms. A CM is also called a loadlock (LL) where wafer loading and unloading, and cassette loading and unloading are performed. A PM is composed of a processing chamber and a chuck that moves up or down to the processing or transfer locations while holding a wafer. In view of wafer transfer, the PMs and CMs are mechanically attached to the TM and wafers move from the TM to one of them or vice versa. In that sense, PMs and CMs are called attached modules (AMs). A wafer transfer job within a cluster tool is performed by the TM in collaboration with one of the AMs (SEMI 1996). As compared with conventional batch processing equipment, the general advantages of cluster tools based on single wafer processing technology are flexibility to produce diverse wafer lots, better yield, reduced contamination, better floor utilization, and less human intervention (Srinivasan 1998). Therefore, cluster tools become mandatory for future semiconductor manufacturing, such as 300 mm wafer fabrication. It is expected that almost 90% of processes will use cluster tools or similar track equipment (Information Network 2002).

Each component module is locally monitored and controlled by a module controller such as a process module controller (PMC), a cassette module controller (CMC), or a transport module controller (TMC). The distributed module controllers are integrated by an inter-module communication network and coordinated by a centralized controller, called a cluster tool controller (CTC) (see figure 1(b)) (SEMI 1996, Shin et al. 2001). The CTC manages the start and completion of wafer processing at each module, wafer transfers by the TMC between the modules, loading and unloading cassettes into the CMs, recipe transfers to the PMCs, detection of abnormal events, and error recovery.

Authors: J.-H. Lee and T.-E. Lee, Korea Advanced Institute of Science and Technology, Department of Industrial Engineering, 373-1, Guseong-dong, Yuseong-gu, Daejeon, 305-701, Republic of Korea. Email: tlee@kaist.ac.kr. J.-H. Park, SunMoon University, Division of Mechanical and Control Engineering, Tangjeong, Asan, Chungnam, 336-708, Republic of Korea.

International Journal of Computer Integrated Manufacturing
ISSN 0951-192X print/ISSN 1362-3052 online © 2004 Taylor & Francis Ltd
http://www.tandf.co.uk/journals
DOI: 10.1080/0951192050000
Modern cluster tools require advanced control functions. First, a cluster tool has diverse logical configurations and wafer flow patterns (Shin et al. 2001). Second, a cluster tool is subject to sporadic exceptional events such as PM breakdowns and failures, therefore requiring fault detection and error recovery. Third, some advanced cluster tools—for instance, for a class of chemical vapour deposition processes—require strict timing constraints on the wafer sojourn times within a PM’s chamber after processing in order to avoid quality problems due to residual gases and heat. Fourth, a cluster tool should support advanced scheduling and control for dual-armed robots as well as single-armed robots. Finally, for simple and easy integration, control applications of a cluster tool should have open and high-level interfaces. Recent advanced process control (APC) functions, fault detection and classification (FDC) for detecting process deviation and run-to-run (R2R) control for adaptive process control based on in-process measurements, also need to be incorporated into the CTC. Such complex distributed control function requirements require advanced module communication functions between the CTC and module controllers, which include high-speed reliable communication, real-time monitoring and control, distributed client–server type communication, high-level messaging and services, complex data transfer, and standardized protocols and services.

Most contemporary cluster tools use TCP/IP and Ethernet for module communication since most module controllers are based on industrial personal computers. Despite their popularity, they have some disadvantages. First, it is difficult to develop high-level communication functions for advanced control, monitoring, and data transfer using low-level TCP/IP services. Second, implementations of such high-level communication functions using TCP/IP services are not standardized. Finally, the response time is not
bounded due to the CSMA/CD (Carrier Sense Multiple Access with Collision Detection) medium access the control of the Ethernet.

SEMI (Semiconductor Equipment and Materials International) defines an application-level communication service and messaging standard, CTMC (Cluster Tool Module Communication), for module communication in cluster tools (SEMI 1996). CTMC specifies application-level services including process management, recipe management, exception handling, and material handling. It does not restrict the lower-level communication protocols and services that enable the high-level application services. Nonetheless, traditional popular SEMI communication standards, SECS-I/II (Semiconductor Equipment Communication Standard-I/II) and HSMS (High Speed Message Specification), are preferable as the communication enablers for CTMC. However, the protocol stacks based on these SEMI communication standards are rather complex, slow, and expensive to implement. Therefore, they are not suitable for complex distributed real-time control. Moreover, the messaging standard, SECS-II, which is defined with stream and function concepts, does not support the object-oriented concept on which CTMC is based. Therefore, we need to consider an alternative implementation method of CTMC. Furthermore, CTMC specification, which is based on conventional material handling concepts, is not enough to deal with control of modern dual-armed cluster tools.

Recently, open fieldbuses have been increasingly adopted for automation and factory communication. In particular, PROFIBUS (PROcess FieldBUS) and other major fieldbuses such as WorldFIP and Foundation Fieldbus have not only high-speed low-level protocols and services for sensor and actuator level communication and control, but also high-level ones for messaging, data communication, and distributed remote control of complex devices. High-level fieldbuses such as PROFIBUS-FMS (Fieldbus Message Specification), subMMS of WorldFIP, and FMS of Foundation Fieldbus are all based on object-oriented concepts and inherited from the precursory MMS (Manufacturing Message Specification), the application layer messaging standard of MAP (Manufacturing Automation Protocol) (ESPRIT 1995, ISO 1990). High-level fieldbuses are expected to be useful for module communication of a cluster tool. First, they provide standardized messaging services based on object-oriented concepts and protocols. We can construct an open communication network for integrating diverse devices and applications. Second, high-level fieldbus services are designed for complex distributed control applications that monitor and coordinate many complex distributed local controllers. Third, they support fast real-time communication for their deterministic response time characteristics. Since they use only the 1st, 2nd, and 7th layers of the OSI reference model, the protocol stacks are simple, light, and efficient. Finally, other general advantages of fieldbuses include high reliability and performance, powerful services, reduction of cabling, and easy diagnosis and maintenance. In spite of these advantages, it is not easy to implement and use PROFIBUS-FMS because traditional factory communication has relied on PLC (Programmable Logic Controller)-based methods and factory automation engineers are not familiar with the concept of communication objects. Moreover, to utilize the full functionality of PROFIBUS-FMS, communication objects should be properly configured. Therefore, it is necessary to provide a PROFIBUS-FMS application method that can be used as a guideline or reference model. We note that low-level fieldbuses such as PROFIBUS-DP (Distributed Peripherals) and LonWorks are already used for sensors and actuator-based low-level process control within the PMs of a cluster tool (Angus 1999).

In this paper, we propose the use of PROFIBUS-FMS, a major high-level fieldbus, for implementing CTMC. We first discuss module communication requirements. We analyse objects and services of CTMC and compare the communication concepts of CTMC and PROFIBUS-FMS. We propose improvements of a key object of CTMC for material transfer and the part transfer protocol to meet the functional requirements of modern dual-armed cluster tools. We also discuss mapping objects and services of CTMC to PROFIBUS-FMS communication objects and services. Finally, we explain how to implement the mappings.

2. Module communication

2.1. CTMC overview

CTMC is a SEMI standard (E38) that addresses an uppermost application-level communication method for module communication in cluster tools (SEMI 1996). It is based on object-oriented concepts and other existing SEMI software standards for general semiconductor manufacturing systems such as material movement management (E32), object services (E39), process management (E40), exception management (E41), recipe management (E42), and event reporting (E53) (SEMI 1995). CTMC specifies information models, application objects, and the associated communication services for CTC, which are necessary for monitoring and coordinating the module controllers. The communication services are defined by specifying interactions between the objects. Major objects defined in CTMC
are PRJob for processing management, TRJob for transfer job management, AtomicTransfer for managing atomic transfers, RecipeExecutor for managing recipes, EventReport for managing events, etc. All objects have a common attribute, ObjType, to represent their object type, and operations, Get and Set, for other objects to access their attributes by SEMI’s another standard, object services (E39).

CTMC is independent of the lower-level communication protocols that transfer the CTMC messages between the module controllers and the CTC. There can be several alternatives for low-level communication enablers for CTMC, including TCP/IP and Ethernet, SECS-I/II/HSMS, a fieldbus, or middleware such as CORBA (Common Object Request Broker Architecture). SECS/GEM (Generic Equipment Model) is a typical communication method between a host (CTC) and devices (that is, module controllers) in general semiconductor manufacturing using SECS-I/II with GEM and VFEI (Virtual Factory Equipment Interface). GEM is an application-level standard that defines generic behaviour of semiconductor equipment (SEMI 1998). It also defines which SECS-II messages should be used, in what situation, and what the resulting activity should be. GEM defines the capabilities of devices such as communication establishment, event notification, remote control, process program management, material movement, etc. VFEI is a messaging interface between a cell control application and equipment drivers in a host application. It can be considered as a message definition inside a host application. GEM and VFEI are application-level standards for semiconductor manufacturing equipment. However, in general, they are not fully supported by vendors and too low-level to be easily used for factory applications. SECS/CTMC uses CTMC, a domain standard for cluster tools, instead of GEM. CORBA (Common Object Request Broker Architecture) presumes high-level interactions between distributed application objects through method invocation between remote objects instead of message transfer. In general, it is neither primarily intended nor popular for real-time communication in a manufacturing environment and is usually based on TCP/IP, although there is a real-time version of CORBA. Compared with message transfer, method invocation has not yet been proven well for real-time manufacturing. Recalling the previously explained disadvantages of TCP/IP and Ethernet, and SECS-I/II/HSMS, a better communication enabler for CTMC implementation is desirable. The requirements of the communication enabler include high-speed real-time messaging capabilities, popularity in manufacturing application, object orientation, advanced high-level communication services for easy implementation of CTMC objects and services, and advanced distributed control functions of complex remote local controllers such as module controllers. The distributed control functions are handling events like task completions or alarms, reading and writing complex variables for process control parameters, downloading files such as recipes or process control programs, commanding module controllers for specific remote control tasks such as starting processing at a PM or starting a move task by a robot, and distributed control or messaging between local controllers. We therefore consider high-level fieldbuses such as PROFIBUS-FMS for fulfilling such requirements. We propose PROFIBUS-FMS/CTMC that replaces SECS with PROFIBUS-FMS (see figure 2(a)). Module communication methods with their protocol stacks are shown in figure 2(b). The layers of SECS and HSMS do not exactly match with the OSI seven layers. The protocol stack of PROFIBUS-FMS/CTMC is light and fast since it is composed of only the 1st (Physical), 2nd (Data Link), and 7th (Application) layers of the OSI reference model, but also guarantees real-time communication.

2.2. PROFIBUS-FMS as an enabler for CTMC

PROFIBUS is one of the major fieldbus protocols. It is a German standard (DIN 19245), a European standard (EN 50170), and now one of the world fieldbus standards (IEC 61158). It has three classes of protocols, PROFIBUS-DP ( Distributed Peripherals) for short and fast Input/Output data exchanges, PROFIBUS-FMS ( Fieldbus Message Specification) for high-level messaging, complex data transfer services, and distributed remote control, and PROFIBUS-PA ( Process Automation) for reliable real-time continuous process control (PNO 1991a, b, c).

PROFIBUS-FMS provides 14 generic communication objects and 39 associated communication services (see figure 3). VFD (Virtual Field Device), PI (Program Invocation), Domain, Event, and Variable are the most important communication objects. A SimpleVariable object is similar to a simple variable of a programming language that can have various types such as integer or real. An Array object consists of a collection of SimpleVariable objects of the same type, but a Record object can be composed of different types of SimpleVariable objects. A VariableList object is a list of addresses of the above variable objects. A Domain object can contain addresses of all communication objects including those of PhysicalAccess objects. A PI object that corresponds to a program process provides a linkage from domains to a program and is composed of a list of addresses of Domain
objects. Usually, the first Domain object of the list is considered as an executable program. An Event object may be connected to a variable object to represent event data. Each communication object has its own associated communication services. For example, we can use services such as Start, Stop, Reset, and Kill to control the behaviour of a PI object. We summarize the relationships among the communication objects in figure 3. A VFD object is an abstract root object that defines and identifies a manufacturing device in the view of a communication partner, such as a cell control application or the communication function of other device (PNO 1991a). All communication objects that are configured within a device’s controller are subordinate to the VFD.

Communication specific information, such as the network addresses of remote stations, connection types that describe the roles of the participating stations and periodicity of the communication, and what services are enabled in the connections, are contained in CR (Communication Relationship) objects. CR objects are also aggregated into a CRL (Communication Relationship List) object. A TransactionObject is created upon receipt of a confirmed service indication and is uniquely related to the corresponding service (PNO 1991b). A PhysicalAccess object describes the access of an actual octet string for arbitrary data, of which address and semantics are known to the user. It is not a standardized communication object and its use depends on the user or programmer. All other commu-
Communication objects are referred by the VFD via an OD (Object Dictionary). An OD is a list of object descriptions, ObjectDescription objects, that have the addresses of the corresponding communication objects.

Figure 4 shows the client–server communication concept based on PROFIBUS-FMS. Usually, a device’s local controller that performs the requested communication services is called a server while the client is a control application at a remote control computer. Communication between the servers is also possible. A VFD and its subordinate communication objects that are listed in the device’s ObjectDictionary object are configured to perform appropriate communication functions or services for the device. The configuration is performed a priori by the device vendor or later by a control application or network engineer. In the server application of the example, a SimpleVariable object, State, is defined to keep and update the state information of the connected device. To identify the current state of the device, the client VFD issues a READ service request to the server. The VFD in the server receives the request (READ.Req) and returns the current value of variable State to the client by sending READ.Rsp, a response message.

Although both CTMC and PROFIBUS-FMS are application-level messaging standards based on object-oriented concepts for distributed control, they have differences. CTMC is intended and specialized for module communication in a cluster tool. Therefore, it has semiconductor manufacturing specific objects such as wafers and recipes. On the other hand, PROFIBUS-FMS has a generic set of communication objects and associated communication services for general manufacturing applications. Therefore, to implement CTMC using PROFIBUS-FMS, appropriate communication objects should be configured for CTC applications that are specified by CTMC. For example, a SimpleVariable object of PROFIBUS-FMS can be configured to represent a TRJob object of CTMC for transferring a wafer from one module to another. A TRJob object has several services to control its behaviour such as start and stop. However, major communication services for a SimpleVariable object are only for reading and writing. Therefore, we should assign some meaningful values to the SimpleVariable object such as writing ‘1’ for starting the transfer job and ‘2’ for stopping the job. Such value assignment or mapping of CTMC services into PROFIBUS-FMS communication objects depends on the users.
or module vendors. It is essential to standardize the mapping and fully utilize the functionality of CTMC objects so that tool integrators or CTC application programmers can easily integrate the module controllers using the standardized PROFIBUS-FMS objects. A profile of PROFIBUS-FMS or a companion standard of MMS for a specific major manufacturing device such as NCs or robots specifies standardized communication objects or services for a device (ISO 1991, 1992). Control application engineers or integrators can easily or consistently use the standardized communication objects or services of the devices without the need of understanding or translating the vendor specific implementation once the device vendors implement the profiles or companion standards for the devices. We therefore propose a mapping or implementation of PROFIBUS-FMS for cluster tools that conforms to CTMC.

3. Analysis and modification of CTMC

We now discuss whether CTMC objects and services are sufficient for modern advanced cluster tools. Modern cluster tools tend to have dual-arms for efficiency, capabilities for dealing with frequent recipe changes and wafer flow pattern switching, reliable error detection and recovery and diverse exception handling capabilities, strict timing control functions for controlling the wafer residency times within PM chambers after processing until unloading, and APC (Advanced Process Control) functions for real-time process fault detection and adaptive process control through in-process measurement. Therefore, the functional requirements for module communication include fast real-time data transfer and messaging, fast transfer of complex data or files including recipe files and process parameters, high-level or application-level messaging services, reliable and efficient support for complex material handling operations, and advanced event monitoring and management. Most of these requirements can be addressed by using fast reliable PROFIBUS-FMS as the communication enabler for CTMC. However, CTMC objects and services are not sufficient for advanced material handling functions for dual-armed cluster tools because CTMC’s material handling model is based on conventional SEMI standards for generic material handling. We therefore need to upgrade TRJob objects and the part transfer protocol (PTP).

Essential CTMC objects for wafer processing are TRJob for wafer transfer and PRJob for managing processing at the PMs. A simple example that shows message exchange sequences using these can be found in the ‘Typical Operating Scenario’ section of the Appendix of the CTMC Standard (SEMI 1996). In this section, we will explain a more complicated operation using a dual-armed robot, called swap, with the corresponding message sequences and object configurations.

Dual-armed cluster tools mostly use swap operations in which the robot unloads a wafer from an AM into an empty arm, and immediately loads the next wafer on the other arm into the AM after rotating the dual opposite-sided arms. Such a swap operation is repeated for the AMs in the sequence of the wafer flow pattern. The use of swap operations simplifies the robot task scheduling and reduces the cycle time (Kim et al. 2003). We will investigate the appropriateness of TRJob object for executing a swap operation by considering the following situation. PM1 has a wafer (001) processed and CM1 has a wafer (002) to be processed at PM1. Then, the CTC asks the TM to first pick up wafer (002) at CM1 and move it to PM1, and then to perform a swap operation that unloads wafer (001) from PM1 into the empty arm, and loads wafer (002) on the other arm into PM1 (after rotating the arms 180°). The TM proceeds to the next swap operation for processing wafer (001) at
PM2. A message exchange sequence and an object configuration for the swap operation are shown in figures 5(a) and 6(a), respectively.

A swap operation using TRJob objects requires four participants. They are the CTC that initiates the transfer job and receives an acknowledgement of job completion, a source AM where the wafer is unloaded, a destination AM where the wafer is loaded, and the TM that transfers the wafer between the AMs. In order to execute a swap operation, the CTC first creates TRJob objects at each participating module shown in figure 6(a) by sending TRJobCreate messages to each module as shown in figure 5(a). For example, at PM1, two TRJob objects, PM_TR1 and PM_TR2, with subordinated AtomicTransfer objects, PM_ATR1 and PM_ATR2, are instantiated. Then, CM1 and PM1 exchange hand-off messages with the TM for real wafer delivery. After creating two TRJob objects, PM_TR1 and PM_TR2, at PM1, PM1 issues a HOReady message to the TM to indicate that it is ready for unloading the wafer from PM1. When the TM finishes the wafer unloading task at CM1, it returns a HOReady message to PM1 to notify that it is ready to perform the unloading operation at PM1. Then, PM1 sends the

<table>
<thead>
<tr>
<th>CTC</th>
<th>TM</th>
<th>CM1</th>
<th>PM1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Transfer Job Setup</strong></td>
<td><strong>Transfer Job Setup</strong></td>
<td><strong>Transfer Job Setup</strong></td>
<td><strong>Transfer Job Setup</strong></td>
</tr>
<tr>
<td>TRJobCreate(Rsp)</td>
<td>TRJobCreate(Rsp)</td>
<td>TRJobCreate(Rsp)</td>
<td>TRJobCreate(Rsp)</td>
</tr>
<tr>
<td>TRJobCreate(Rsp) (TM, TR1)</td>
<td>TRJobCreate(Rsp) (TM, TR2)</td>
<td>TRJobCreate(Rsp) (PM, TR1)</td>
<td>TRJobCreate(Rsp) (PM, TR2)</td>
</tr>
<tr>
<td><strong>CM1 Handoff</strong></td>
<td><strong>PM1 Handoff</strong></td>
<td><strong>TM to CM1 Handoff</strong></td>
<td><strong>TM to PM1 Handoff</strong></td>
</tr>
<tr>
<td>TRJobCompleted(Rsp)</td>
<td>TRJobCompleted(Rsp)</td>
<td>TRJobCompleted(Rsp)</td>
<td>TRJobCompleted(Rsp)</td>
</tr>
<tr>
<td>HOReady(Rsp)</td>
<td>HOReady(Rsp)</td>
<td>HOReady(Rsp)</td>
<td>HOReady(Rsp)</td>
</tr>
<tr>
<td>HOPick(Rsp)</td>
<td>HOPick(Rsp)</td>
<td>HOPick(Rsp)</td>
<td>HOPick(Rsp)</td>
</tr>
<tr>
<td>HOVerify(Rsp)</td>
<td>HOVerify(Rsp)</td>
<td>HOVerify(Rsp)</td>
<td>HOVerify(Rsp)</td>
</tr>
<tr>
<td><strong>TM Setup</strong></td>
<td><strong>CTC Setup</strong></td>
<td><strong>CM1 Setup</strong></td>
<td><strong>PM1 Setup</strong></td>
</tr>
<tr>
<td>TRJobCreate(Rsp)</td>
<td>TRJobCreate(Rsp)</td>
<td>TRJobCreate(Rsp)</td>
<td>TRJobCreate(Rsp)</td>
</tr>
<tr>
<td>TRJobCreate(Rsp) (TM, TR1)</td>
<td>TRJobCreate(Rsp) (TM, TR2)</td>
<td>TRJobCreate(Rsp) (PM, TR1)</td>
<td>TRJobCreate(Rsp) (PM, TR2)</td>
</tr>
</tbody>
</table>
TM a HOPick message to ask unloading wafer (001) from PM1. The TM sends PM1 a HOVerify message to verify whether the unloading job is completed on both sides. After receiving the HOVerify message, PM1 sends a TRJobComplete message to the CTC and hence the transfer job PM_TR1 at PM1 is now completed. Loading wafer (002) at PM1 is performed in the same way. Unloading from or loading to an AM are called pick and place, respectively. A transfer job is performed by a TRJob object and its associated two AtomicTransfer objects for pick and place. The macro-level messages between the CTC, AMs, and the TM for creating TRJob objects are followed by the micro-level messages between the TM and AMs, which are performed by the AtomicTransfer objects. An AtomicTransfer object performs an atomic wafer transfer for changing the possession of a wafer, for instance, from PM1 to the TM.

For a single swap task, many TRJob objects are instantiated at several modules and interact with each other in a complicated manner. Moreover, TRJob objects with related AtomicTransfer objects instantiated at the TM are not executed sequentially. The TRJob and AtomicTransfer objects at the TM are executed in parallel: TM_ATR1-1 (TM_TR1 starts), TM_ATR1-2 (TM_TR1 pauses and TM_TR2 starts), and TM_ATR1-2 (TM_TR2 pauses and TM_TR1 resumes and finishes). In addition, as explained later, it is complicated to implement the TRJob objects and AtomicTransfer objects using PROFIBUS-FMS services. Such complications are because a transfer job is defined based on a

<table>
<thead>
<tr>
<th>CTC</th>
<th>TM</th>
<th>CM1</th>
<th>PM1</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRJobCreate(pick), Req</td>
<td>TRJobCreate, Resp</td>
<td>TRJobComplete, Req</td>
<td></td>
</tr>
<tr>
<td>TRJobCreate, Req</td>
<td>TRJobComplete, Resp</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5. CTMC message exchange sequences for the swap operation. (a) Using original TRJob objects. (b) Using modified TRJob objects.
material-centred view, where a material handling operation is specified by three parameters: material to be transferred, the source where the material resides, and the destination where the material is to be placed next. For example, a transfer job request is represented as TRJobCreate.Req(MID = "Wafer1", TRSourceAmID = "CM", TRDestinationAmID = "PM1") (SEMI 1996). Although such a material-centred approach is useful for most automated material handling system operations, it is not appropriate for implementing a swap operation at a dual-armed cluster tool. Since one of the dual arms can be used as a temporary buffer during a swap operation, a swap operation performs elementary material handling operations in the sequence of pick - pick - place - pick. This sequence is quite different from a usual material handling operation sequence, pick - place - pick - place. To resolve this problem, we define a material handling operation by a task-centred view instead of the material-centred view. In other words, a material handling operation should be specified by a task type to be executed, a material to be handled, and a module where the task is performed. The task types are pick, place and swap. Such task type definitions are often used by tool vendors for scheduling dual-armed cluster tools (Shin et al. 2001). Table 1 shows the proposed task-centred modification of the TRJob object model.

We modify the TRJob object model and eliminate the AtomicTransfer objects for simplicity and efficiency. Among the attributes, TRSwapObjName may not be used or need not be specified depending on the task type. If the task type is swap, it indicates the identifier of the wafer to be swapped. If the task type is pick or place, the attribute need not be used. Such a conditional attribute is marked by symbol C in a round bracket. The message sequence for the swap operation using the modified TRJob object is shown in figure 5(b). It is different from the message sequence for the wafer transfer operation using the original TRJob object. First, the AtomicTransfer objects are no
longer used (see figures 6(b)). Although they are useful for tracing the detailed status information of the transfer operation, the object composition and related interactions using the AtomicTransfer objects are complex. Second, the number of participants and TRJob objects of the transfer operation is reduced from four to three. For example, in the original swap operation shown in figures 5(a), the CTC sets up TRJob objects in three modules, the TM, CM1, and PM1. However, only two TRJob objects are configured at the same time for any wafer transfer operation in figure 5(b). When the pick operation at CM1 is performed, two TRJob objects, CM_TR1 at CM1 and TM_TR1 at the TM, exist, and during execution of the swap operation at PM1, another two TRJob objects, TM_TR2 at the TM and PM_TR1 at PM1, are instantiated (see figures 6(b)). The message sequence and interactions between the modules for the wafer transfer job become more simple and efficient.

The second improvement for CTMC should be made for the message sequence between the modules for material movement. Gong (1998) defines the message sequence for automated flow line systems, named a Part Transfer Protocol (PTP). A part transfer is either for machine loading or unloading. Therefore, depending on the initiative of the activity, the PTPs are classified into four cases, machine loading initiated by a machine, machine unloading initiated by a transporter, machine loading initiated by a machine, and machine loading initiated by a transporter. In CTMC, the second and fourth cases are generally used. That is, the PTPs used in CTMC are initiated by a transporter, TM. A typical cluster tool has only one TM, which is a unique transport partner of other AMs.

Therefore, for usual purposes, it suffices to use the PTPs that are initiated by the transporter, TM. However, we propose additional types of PTPs, called controller-initiated PTPs for the following reasons. First, some recent cluster tools or similar track equipment have multiple TMs, and hence the material handling partner of AMs in the PTP may not be unique. Therefore, the transporter-initiated PTPs are not appropriate. Second, we can model interactions between the modules themselves using method invocations between objects in the CTC application, which can be conveniently used for implementing a complex control logic. Finally, deliberate timing control can be achieved by controller-initiated PTPs. To meet strict in-chamber wafer residency time constraints for advanced processes such as low pressure chemical vapour deposition, it is necessary to appropriately control timings of the TM’s transfer jobs. Controller-initiated PTPs are essential for such timing control because the CTC collects all events and timing information and can make complex timing control decisions (Kim et al. 2003). Figure 7 illustrates a transporter-initiated PTP and a controller-initiated PTP.

The PTP examples show the CTMC messages. In each message sequence, the messages in a shaded area correspond to a PTP. Before the PTP is performed, the CTC creates TRJob objects in the TM and the PM using TRJobCreate service. The created TRJob objects start their processes and notify the CTC using the TRJob-Started service of the event. In the transporter-initiated PTP in figure 7(a), the PM sends a HOReady message to the TM to inform that the PM is ready to perform the wafer transfer job. After receiving the HOReady message from the PM, the TM determines when the wafer transfer job begins. Then, the HOVerify message is sent from the PM to the TM and physical wafer handling is performed by the TM. After the wafer is handed over to the TM, a verification message, HOVerify, is exchanged between the modules and a message indicating the end of TRJob, TRJobComplete, is sent to the CTC.

In a controller-initiated PTP, the PM and the TM send HOReady messages not to their transfer partner modules but to the CTC. Then the CTC determines the start of the wafer transfer job and sends HOPick messages to the TM and the PM. After the TM executes the order, the CTC verifies whether the wafer transfer job is successfully completed. In this PTP, the CTC coordinates all the related activities of the modules. There is no direct interaction among the modules. If the supervisory control function of the CTC is well designed, the controller-initiated PTP can perform the complex job efficiently.
SEMI’s material movement management standard defines a sequence of messages called *message flow via host* (SEMI 1998). However, it is different from the proposed controller-initiated PTP. In a message flow via host, each module controller sends hand-off messages to the transfer job partner (that is, another module controller), and the messages do not go directly to the partner but are sent to the host first and then to the partners. A message flow via host is used when there is no direct link between the two module controllers. The message exchange sequence is initiated by a module controller called the *primary partner*. In a controller-initiated PTP, all message exchanges are initiated by the host (that is, the CTC in a cluster tool).

By using PROFIBUS-FMS as the communication enabler of CTMC, we obtain another advanced function that changes the control program at the modules as needed. Modern semiconductor manufacturing requires the production flexibility of changing the lots or recipes frequently. Therefore, it is necessary to adjust the wafer flow pattern and the parameters of the process control programs at the PMs, and often completely switch the process control programs to others. A control program file or a control parameter data file should be downloaded and executed. CTMC does not support such file downloading and controlling of the program processes. PROFIBUS-FMS provides file download services and program process control services by Domain objects and PI (Program Invocation) objects, respectively.

Figure 7. PTP examples for different initiators. (a) Transporter-initiated PTP. (b) Controller-initiated PTP.

Figure 8. CMC Object models. (a) CMC Object model. (b) Modified CMC Object model.
4. Mapping CTMC objects to PROFIBUS-FMS communication objects

Based on the modified TRJob object, we accordingly modify the object models of CTMC for the module controllers. We focus on the essential objects, PRJob, TRJob, EventReport, Wafer, and Cassette, which are affected by the TRJob object modification. The TRJob object for a CMC (Cassette Module Controller) performs loading and unloading a wafer at a CM, or loading and unloading a cassette at a CM in collaboration with an external transfer device such as an automatic guided vehicle or a human operator. Since a TRJob object is modified to handle all generic transfer operations, pick and place as well as swap, we remove the atomic transfer objects. Figures 8(a) and 8(b) show the original CMC object model in CTMC and the modified CMC object model, respectively. An EventReporting object monitors exceptional events such as module or operation failures. A Device object is specified depending on the device that is controlled by the module controller, such as a robot arm or a slot valve of the TMC, and a chuck or a PM chamber of a PMC. For example, we provide Domain objects that have the addresses of some Variable objects that represent device states, which are usually configured by device manufacturers (ISO 1992).

The object models for a PMC and the TMC are also modified similarly in order to incorporate the modified TRJob object as shown in figures 9(a) and 9(b), respectively. A PM receives a wafer from the TM, processes the wafer at a chamber device, and returns the wafer to the TM. When the wafer recipe is changed, it downloads a new recipe that contains the data and device capabilities necessary for processing the wafer type, or the whole control program. A wafer processing job is performed by a PRJob object. The Device object in the TMC Object model can be defined for a wafer handling device such as a robot arm. Figure 10(a) shows the attributes and the state transition model of a Wafer object in CTMC (SEMI 1996). Attribute OccModule indicates the module where the wafer currently resides. A Cassette object has multiple Slot objects, each of which can keep a Wafer object, as shown in figure 11.

The basic concepts of CTMC objects and PROFIBUS-FMS objects are quite different. The SEMI software standards on which CTMC is based are organized by each application function. For example, a TRJob object

![Diagram of PMC and TMC Object models.](image-url)
is defined for material movement management (E32), a PRJob object for process management (E40), a Recipe object for recipe management (E42), and an EventReporting object for event reporting (E53). They cover the CTC application functions for interacting with the modules. However, the objects of PROFIBUS-FMS are not designed for the functions of a specific application, but defined as generic components for distributed manufacturing applications. They include PI objects for defining and running programs on a computer system, Domain objects for transferring large or bulk data or process control programs, and SimpleVariable objects for reading and writing parameters or simple messages. Therefore, it is necessary to fill the gap between CTMC and PROFIBUS-FMS.

In order easily to integrate CTMC applications with PROFIBUS-FMS communication functions, it is essential to map full CTMC objects to PROFIBUS-FMS objects. We first investigate the possibility of one-to-one mapping. In general, one PROFIBUS-FMS object is not enough for representing the whole semantics of any CTMC object. However, each Recipe object and EventReporting object can be mapped into only one PROFIBUS-FMS object. A Recipe object corresponds to a Domain object of PROFIBUS-FMS that models a parameter data file or a control program that may contain another PROFIBUS-FMS objects. Recipe management standard (E42) is too complex for semiconductor equipment vendors to follow faithfully. However, the recipe transfer function, which is the essential function for wafer processing in a cluster tool, can be easily implemented by domain upload and download services of a Domain object in PROFIBUS-FMS. Although an EventReporting object of CTMC has more functionalities, it can be well mapped into an Event object of PROFIBUS-FMS because they have basically identical purposes.

PROFIBUS-FMS objects have predefined meanings and it is not easy to expand their definitions. We cannot add or modify attributes or services. A strategy for expanding a PROFIBUS-FMS object definition is to use a generic object, a SimpleVariable object, for defining...
the desired attributes and associating them with the original object, for instance, PI. A CTMC object definition consists of two parts, attributes and services. Since it is hard to represent a whole object with a PROFIBUS-FMS object, the attributes of a CTMC object are modelled by variable objects of PROFIBUS-FMS and the variable objects are associated with the main PROFIBUS-FMS object to which the CTMC object is mapped. The CTMC services are then assigned to the mapped PROFIBUS-FMS objects. For example, three different SimpleVariable objects of PROFIBUS-FMS are used for modelling three attributes of a Wafer object of CTMC as shown in figure 10(a).

The proposed mapping guidelines are summarized in table 2. An attribute of a CTMC object, for instance, ObjType attribute and PRJobID attribute of a PRJob object, can be expressed by a SimpleVariable object of PROFIBUS-FMS. A CTMC object that has more than one such attribute can be expressed by a PROFIBUS-FMS container object such as a Record object, a VariableList object, or a Domain object. A static object, which is once created and persists along the communication application, can be mapped into a Record object. A dynamic object, which can be created and deleted at any time during communication, can be mapped to a VariableList object. If the size of the object is too large to be contained in a Record object or a VariableList object, we can use a Domain object that can include arbitrary objects and data. If one wishes to access the whole object and uses Record objects or VariableList objects, they can be easily accessed by their simple Read or Write services. However, when a Domain object is used, it must be downloaded by a Domain download service that is accompanied by a rather complex message sequence. Therefore, using a Record object or a VariableList object is recommended rather than a Domain object as often as possible.

The mapping guidelines in table 2, even though simple, utilize almost full functionalities of PROFIBUS-FMS. Some CTMC objects such as TRJob and PRJob contain an implicit feature, process, which is not shown in the object definition. Such a process can be implemented by a PI object or Variable objects with proprietary definitions. For example, to control the behaviour of the process of a TRJob object, we can add three SimpleVariable objects, SV_Start, SV_Stop, and SV_Abort. If we write a predefined value like ‘1’ to SV_Abort, the process of TRJob object aborts as if the TRJobCommand service with abort parameter is called. Instead of using the three SimpleVariable objects, we can let only one SimpleVariable perform such a job. Write service with parameter ‘1’ indicates process start, ‘2’ stop, and ‘3’ abort, etc. However, although this method is simple and adaptable for various applications, it is hard to standardize the variables and related parameters. A better method for controlling a process is to use a PI object. In this case, since there is no explicit way of connecting PI objects with variable objects that indicate the attributes of the corresponding CTMC objects as shown in figure 3, we define a Domain object that contains the variable objects that are referenced from the PI objects.

When an object or an attribute of CTMC is mapped onto a PROFIBUS-FMS object, we add an appropriate prefix to the PROFIBUS-FMS object to represent the object type. Prefixes \( SV \), \( R \), \( D \), and \( P \) indicate a SimpleVariable object, a Record object, a Domain object, and a PI object, respectively. We first map a Wafer object. Mapping of a Wafer object to PROFIBUS-FMS objects is illustrated in figure 12.

Each attribute of a Wafer object is mapped to a SimpleVariable object of PROFIBUS-FMS. For example, the attribute WaferID is mapped to a SimpleVariable object, SV_WaferID. There are two additional attributes that are subordinated to the WaferState attribute, SV_ExecSubState and SV_ProcSubState. Since it is difficult to represent the wafer state in figure 10(b) using only a SimpleVariable object, we represent the wafer state using three SimpleVariable objects. SV_ExeSubState and SV_ProcSubState show the substates of state Executing and state Process, respectively. The two added conditional variable objects have meaningful

<table>
<thead>
<tr>
<th>CTMC</th>
<th>PROFIBUS-FMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes of objects</td>
<td>SimpleVariable object</td>
</tr>
<tr>
<td>An object</td>
<td>Record object (if static)</td>
</tr>
<tr>
<td>Process</td>
<td>VariableList object (if dynamic)</td>
</tr>
<tr>
<td>Recipe</td>
<td>Domain object (if the size is too large)</td>
</tr>
<tr>
<td>Event Report object</td>
<td>PI object</td>
</tr>
<tr>
<td>Operations of objects</td>
<td>Services of the mapped objects</td>
</tr>
</tbody>
</table>

Table 2. Guidelines for mapping from CTMC to PROFIBUS-FMS.
values when the current state is *In Cluster Tool*. We have five SimpleVariable objects for the attributes of the Wafer object and define a Record object, \( R\_\text{Wafer} \), for the Wafer object itself. The Record object contains all SimpleVariable objects. To refer the Wafer object in a PI object, we define an additional Domain object, \( D\_\text{Wafer} \), that includes the Record object.

Figure 13 explains how to map a Cassette object. As in the Wafer object, each attribute of a Slot object is mapped to a SimpleVariable object, and a Record object, \( R\_\text{Slot} \), is defined. We can define an additional Domain object, \( D\_\text{Cassette} \), that contains all \( R\_\text{Slot} \) objects referred by a PI object. Such a Domain may include detailed process and quality information in individual wafers for APC.

A PROFIBUS-FMS model for a TRJob object is shown in figure 14. As in other objects, the attributes are mapped to SimpleVariable objects. All SimpleVariable objects are grouped into a Record, \( R\_\text{TRJob} \). The \( R\_\text{TRJob} \) object may be included in a Domain object, \( D\_\text{TRJob} \), to be referred from a PI object, \( P\_\text{TRJob} \), which also can be configured to execute a transfer job.

The TRJob object shown in figure 14 is the modified version of the original TRJob object of CTMC. When the modified TRJob object is used, the TM has only one TRJob object at any time. However, when the original TRJob object definition is used, the TM should have two TRJob objects with four subordinated AtomicTransfer objects to implement a swap operation as shown in figure 15. Moreover, they even form an object hierarchy that results in very complex configurations of PROFIBUS-FMS communication objects. Due to the modification of the TRJob object definition, not only the representation of a transfer job but also the configuration of the PROFIBUS-FMS communication objects are significantly simplified.

A PROFIBUS-FMS object model for a PRJob object is shown in figure 16. A PRJob object is mapped to a PI object, \( P\_\text{PRJob} \). The \( P\_\text{PRJob} \) object is associated with the Domain objects that represent devices (\( D\_\text{Device} \)), recipes (\( D\_\text{Recipe} \)), and a SimpleVariable object (\( SV\_\text{PRState} \)) for notification of state changes.

5. CTMC service mapping

When the objects of CTMC are mapped to PROFIBUS-FMS communication objects, the services associated with the CTMC objects should be executed by the services of the mapped PROFIBUS-FMS communication objects. A Wafer object and a Cassette object are mainly mapped to Variable objects. Therefore, the services to access them are implemented by Read and Write services for Variable objects in PROFIBUS-FMS. In this section, we discuss how to implement the services of complicated objects, a TRJob object and a PRJob object.
Material movement messaging services for the TRJob objects are TRJobCreate for creating a TRJob object, TRJobCommand for aborting the TRJob process, TRJobAlert for notifying state changes of the TRJob object, and TRJobComplete for notifying the job completion (SEMI 1996). These CTMC services are mapped to PROFIBUS-FMS services as specified in table 3. For instance, to execute a single TRJobCreate service of CTMC, three consecutive PROFIBUS-FMS services, Write, Create, and Start are used. To implement a TRJobCreate service of CTMC, after the transfer order parameters are set, a PI object for the TRJob object is created by the Create service of PROFIBUS-FMS. While the TRJob object starts the transfer job automatically, the PI object should be commissioned by another service, Start.

Processing message services for PRJob objects are PRJobCreate for creating a PRJob object, PRJobCommand for aborting the PRJob process, PRJobAlert for notifying state changes at the PRJob object, and PRJobComplete for notifying the service completion (SEMI 1996). These CTMC services are mapped to PROFIBUS-FMS services as specified in table 4. For instance, to execute a single PRJobCreate service of CTMC, three consecutive PROFIBUS-FMS services, Write, Create, and Start are used.

To implement a TRJobCreate service of CTMC, after the transfer order parameters are set, a PI object for the TRJob object is created by the Create service of PROFIBUS-FMS. While the TRJob object starts the transfer job automatically, the PI object should be commissioned by another service, Start.

Processing message services for PRJob objects are PRJobCreate for creating a PRJob object, PRJobCommand for aborting the PRJob process, PRJobAlert for notifying state changes at the PRJob object, and PRJobComplete for notifying the service completion (SEMI 1996). These services are mapped as shown in figure 4.

Get service of SEMI’s object services standard (E39) for reading attributes of an object is mapped to Read service of PROFIBUS-FMS. In CTMC, the behaviour of TRJob or PRJob objects is controlled by TRJobCommand or PRJobCommand service with an appropriate parameter such as start, stop, or resume. In PROFIBUS-FMS, there are distinct services for the corresponding PI object, Start, Stop, and Resume that correspond to the parameters. Event and Recipe services also can be implemented similarly.

The CTMC message sequence for a swap operation in figure 5(b) is implemented by the mapped PROFIBUS-FMS services as in figure 17. For expositional convenience, only the messages at PM1 are presented and all response messages are omitted. Since the message sequence is based on the modified TRJob object and the controller-initiated PTP, it is different from the ordinary CTMC message sequence in the standard (SEMI 1996).

We map CTMC objects and related services to those of PROFIBUS-FMS and propose the communication objects. By using these communication objects and services, the communication among modules of a cluster tool can be standardized and have improved efficiency.

6. PROFIBUS-FMS implementation

Lee and Lee (2003) propose a distributed controller architecture based on PROFIBUS-FMS, and the notion of service objects that map the commands from the control application into appropriate communication services, for instance, PROFIBUS-FMS services. Such service objects are useful for mapping CTMC application services to PROFIBUS-FMS communication services. The overall communication and application architectures based on such concepts are proposed as in figure 18.

GTC and module controllers are typically based on industrial personal computers. For the network interface with PROFIBUS-FMS, they have PROFIBUS-FMS network interface cards that support full objects and services of PROFIBUS-FMS. A commercial example is PROFIboard of Softing with an API (Application Programming Interface) library. However, although the design of PROFIBUS-FMS is based on object-oriented concepts, most commercial implementations of PROFIBUS-FMS are rather based on conventional function calls of C programming language, which is far from object-orientation (Lee and Lee 2003). The PROFIBUS-FMS communication services tend to be
Table 3. Mapping of CTMC services for TRJob objects to PROFIBUS-FMS services.

<table>
<thead>
<tr>
<th>CTMC services</th>
<th>PROFIBUS-FMS services</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRJobCreate(WAFER, ‘Wafer011’, ‘Port01’, PUT);</td>
<td>Write(TRJob, ‘Wafer011’, ‘Port01’, PUT);</td>
</tr>
<tr>
<td></td>
<td>Create(1, ‘PM1TRJob’, ‘TRJobDomain’);</td>
</tr>
<tr>
<td></td>
<td>Start(‘PM1TRJob’); (if TRJob object starts automatically)</td>
</tr>
<tr>
<td>TRJobCommand(Abort);</td>
<td>Start, Stop, Resume, Reset, and Kill services of PI object</td>
</tr>
<tr>
<td>TRJobAlert(TRState);</td>
<td>InformationReport(SV_TRState);</td>
</tr>
<tr>
<td>TRJobComplete</td>
<td>InformationReport(SV_TRState);</td>
</tr>
</tbody>
</table>

Figure 17. Messaging for a swap operation.

Figure 18. Architecture of CTMC implementation based on PROFIBUS-FMS.
implemented not as methods that are encapsulated into the associated objects but separately from the associated objects. With such functional implementations, it is hard to manage multiple services.

To resolve this problem, Lee and Lee (2003) propose a new object, ServiceObject (SO). It is an object-oriented API for application objects to handle service requests and manage their contexts. By using them, we can define a customized ServiceObject that can handle a group of services that are frequently used together. CTMC messages can be considered as internal messages of CTC that application objects issue to the ServiceObjects which translate the CTMC service requests into a series of PROFIBUS-FMS services by predefined message sequence models. In module controllers, PROFIBUS-FMS communication objects that implement CTMC objects receive the service requests, process them, and return appropriate responses by the application objects in the control module hardware.

A message sequence example using a ServiceObject is illustrated in figure 19. An application object that wishes the TM to perform a swap operation at PM1 issues a service request to the corresponding ServiceObject using CTMC TRJobCreate service. The ServiceObject translates the service request into a series of PROFIBUS-FMS services and sends them to PROFIBUS-FMS objects that implement the corresponding TRJob object in the TMC. In figure 19, the messages between the application object and the ServiceObject are CTMC messages within the CTC. The PROFIBUS-FMS messages occur between the ServiceObject of the CTC and the PROFIBUS-FMS communication objects of the TMC. Therefore, the CTMC that supports the CTC can be easily integrated into the proposed application framework by only changing the messaging partner of application objects to ServiceObjects. This improves the portability and applicability of the CTC.

7. Final remarks

We have proposed a high-level fieldbus-based implementation of CTMC specification on communication services between the modules in a cluster tool. In order to meet the control requirements of modern advanced cluster tools, we improved TRJob objects and the part transfer protocol. We suggested mappings from CTMC objects and services to PROFIBUS-FMS
objects and services. We discussed the similarities and differences between CTMC and PROFIBUS-FMS models. Our proposed mapping and implementation models can be a reference for further implementation or for developing a CTMC-based profile for PROFIBUS-FMS-based implementation of cluster tool module communication. Our proposed methods will be useful for system engineers in module vendors, tool vendors or integrators, and/or even wafer fabricators, who are involved in design and development of control applications for cluster tools or similar integrated semiconductor manufacturing equipment.

Advanced functions such as APC and remote monitoring and control that are being required for modern cluster tools should be further examined and additional objects and services need to be defined.

References


SEMI, 1995, SEMI E38-1.95, Communications Environment HSMS/SECS-II for Cluster Tool Module Communications.

SEMI, 1996, SEMI E38-1.96, Cluster Tool Module Communication (CTMC).

