#	Current State	Trigger	Next State	Action	Comment
				by the instance of MaterialContainer	transitions to Completed.
10	InTransit	makeProcessing()	Processing	DurableLocation ChangedEvent Published by the instance of MaterialContainer	Container has reached destination at process equipment. TransportJob state transitions to Completed.
11	Processing	makeInTransit()	InTransit	DurableLocation ChangedEvent Published by the instance of MaterialContainer	Container has left process equipment. TransportJob state transitions to Executing.
12	Processing	makeManual Control()	ManualControl	DurableLocation ChangedEvent Published by the instance of MaterialContainer	Container has left control of process equipment (not via TransportJob).
13	ManualControl	makeProcessing()	Processing	DurableLocation ChangedEvent Published by the instance of MaterialContainer	Container has entered control of process equipment (not via TransportJob).
14	InTransit	makeManual Control()	ManualControl	DurableLocation ChangedEvent Published by the instance of MaterialContainer	Container has left the control of the TransportJob via a manual output port. TransportJob state transitions to Aborted.
15	ManualControl	makeInTransit()	InTransit	DurableLocation ChangedEvent Published by the instance of MaterialContainer	Container has entered the AMHS via a manual input port. TransportJob state transitions to Executing.
16	ManualControl	none	N/A	none	MaterialContainer instance has been removed.

6.2.4 *Machine Abstract Interface Group (subset)*

This section includes only those interfaces and specific operations from the Machine Abstract Interface Group that are referenced within this component specification. These interfaces are provided here for referential integrity of this specification and are not intended to remain as part of this component. Further, this is not intended to be a comprehensive treatment of the required interfaces, but only the subset needed for reference until the full specification is adopted through a subsequent ballot. When that occurs, this specification can be updated to reference the interfaces directly rather than replicate them here.

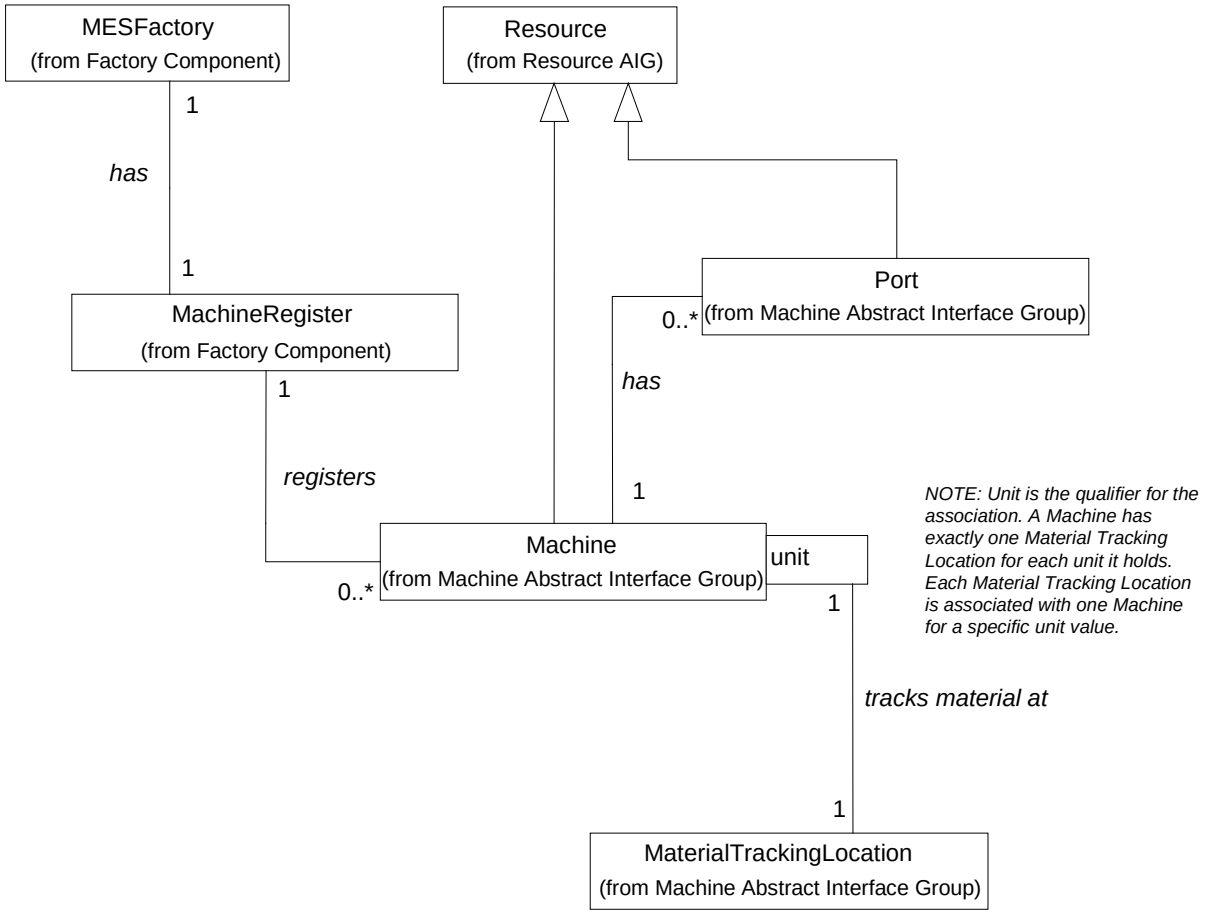

Figure 6
Machine Abstract Interface Group (subset) Information Model

6.2.4.1 *Machine Interface (subset)*

Module: MachineAIG

Interface: Machine

Inherited Interface: Resource

```
interface Machine : AbstractIF::Resource {
```

Description: This abstract interface representing the concept of a piece of equipment in the factory. A Machine establishes the identity of the physical equipment existing in the factory for reference within an MES context. The Machine also maintains one MaterialTrackingLocation for each Unit defined to classify the Material processed, used, or stored by the Machine.

Type Definitions:

```
typedef sequence <Machine> MachineSequence;

typedef sequence <MaterialTrackingLocation> MaterialTrackingLocationSequence;
```

```
typedef sequence <Port> PortSequence;

enum MachineType {
     FixedBufferProductionMachine,
     InternalBufferProductionMachine,
     StorageMachine,
     TransportMachine };
```

Exceptions:

/* Port indicated for receiving material was invalid. */

```
exception InvalidPortSignal { };
```

/* Material specified for unloading was invalid. */

```
exception InvalidMaterialSignal { };
```

Published Events:

/* An E10 State change has occurred in the Machine that the Factory needs to know. */

MachineResourceE10StateChangedEvent

Provided Services:

/* Set the MachineType for this machine. */

```
void setMachineType (
     in MachineType aMachineType )
     raises (Global::FrameworkErrorSignal);
```

/* Return the MachineType for this machine. */

```
MachineType getMachineType ( )
     raises (Global::FrameworkErrorSignal);
```

/* Set up a MaterialTrackingLocation for this Machine to track a particular material Unit characterizing its contents. A Machine may have more than one MaterialTrackingLocation as long as each is tracking a different Unit. The value of Unit is used with corresponding attributes of the material to allocate the tracked material to the correct MaterialTrackingLocation for a Machine. */

```
void setMaterialTrackingLocation (
     in MaterialTrackingLocation aMaterialTrackingLocation,
     in string aUnitToTrack)
     raises (Global::FrameworkErrorSignal);
```

/* Get all MaterialTrackingLocations for this Machine. There is one and only one MaterialTrackingLocation per Unit used in the Machine. */

```
MaterialTrackingLocationSequence getAllMaterialTrackingLocations ( )
     raises (Global::FrameworkErrorSignal);
```

/* Get the MaterialTrackingLocation for the Unit specified. */

```
MaterialTrackingLocation getMaterialTrackingLocation (
     in string aUnit)
     raises (Global::FrameworkErrorSignal);
```

/* Request a Machine to reserve a port to load material. This may involve communication between the Port and the Machine Objects, but these need not be public (or may involve communication between the Machine and the Equipment). If requestedPort parameter left null, machine chooses the requestedPort. Returns Null if Machine cannot or refuses to comply. */

```
Port reservePortForTransferTo (
     in DurablesManagement::MaterialContainer aContainer,
     in Port requestedPort )
     raises (Global::FrameworkErrorSignal,
          InvalidPortSignal);
```

/* Request a Machine to reserve a port to load two or more MaterialContainers. All MaterialContainers in the sequence must be loaded before the Machine can accept another port reservation request. The MaterialContainers specified in the MaterialContainerSequence may be loaded in any order. If requestedPort parameter left null, machine chooses the requestedPort. Returns Null if Machine cannot or refuses to comply. */

```
Port reservePortForBatchTransferTo (
     in DurablesManagement::MaterialContainerSequence aContainerGroup,
     in Port requestedPort )
     raises (Global::FrameworkErrorSignal,
          InvalidPortSignal);
```

/* Request a Machine to cancel a reservation previously established for a transfer to a port. */

```
void cancelPortReservationForTransferTo (
     in DurablesManagement::MaterialContainer aContainer,
     in Port requestedPort )
     raises (Global::FrameworkErrorSignal,
          InvalidPortSignal);
```

/* Inform a Machine that a transfer to or from a port has been completed. This may involve communication between the Port and the Machine Objects, but these need not be public (or may involve communication between the Machine and the Equipment). */

```
void transferComplete (in DurablesManagement::MaterialContainer aContainer,
     in Port reservedPort )
     raises (Global::FrameworkErrorSignal,
          InvalidPortSignal);
```

/* Request a Machine to prepare to unload material. Returns the Port on which the material will be unloaded or Null if the Machine cannot or refuses to comply. */

```
Port containerOut (
     in DurablesManagement::MaterialContainer aContainer,
     in Port requestedPort )
     raises (Global::FrameworkErrorSignal,
          InvalidMaterialSignal);
```

/* Returns a list of all Ports for the machine. */

```
PortSequence allPorts ( )
     raises (Global::FrameworkErrorSignal);
```

/* Returns the list of all Areas that this Machine is in. */

```
AreaSequence allAreas( )
     raises (Global::FrameworkErrorSignal);

}; // Machine
```

6.2.4.2 *MaterialTrackingLocation Interface*

Module: MachineAIG

Interface: MaterialTrackingLocation

Inherited Interface: OwnedEntity

```
interface MaterialTrackingLocation : AbstractIF::OwnedEntity {
```

Description: A MaterialTrackingLocation is a place where Material may be held. The "type" of the location is based on the classification Unit of Material that the location can hold (e.g., locations to hold wafers have a "Wafer" Unit and can only hold this Material type). Each machine has one MaterialTrackingLocation for each material Unit. The MaterialTrackingLocation also maintains a capacity.

Exceptions:

/* Material specified in the call is not a valid Material for this MaterialTrackingLocation. */

```
exception InvalidMaterialSignal { };
```

/* Intent to assign more Material to the MaterialTrackingLocation failed because it is full. */

```
exception MaterialTrackingLocationFullSignal { };
```

/* Unit value is not valid for the MaterialTrackingLocation. */

```
exception InvalidUnitSignal { };
```

Published Events:

/* A Material Tracking Location has reached its capacity. */

```
MaterialTrackingLocationFullEvent
```

/* A Material Tracking Location that was previously Full now has available capacity. */

```
MaterialTrackingLocationNotFullEvent
```

Provided Services:

/* Return the Material at the MaterialTrackingLocation, return nil if the location is empty. */

```
AbstractIF::MaterialSequence allMaterialHeld ( )
     raises (Global::FrameworkErrorSignal);
```

/* Get and set the unique identifier for the MaterialTrackingLocation. */

```
string getIdentifier ( )
     raises (Global::FrameworkErrorSignal);

void setIdentifier (
     in string identifier)
     raises (Global::FrameworkErrorSignal,
          Global::SetValueOutOfRangeSignal,
          Global::DuplicateIdentifierSignal);
```

/* Return the Machine that holds this MaterialTrackingLocation. */

```
Machine getMachine ( )
     raises (Global::FrameworkErrorSignal);
```

/* Set the material Unit that can be held in the MaterialTrackingLocation. Example: If the MaterialTrackingLocation is to hold wafers, the Unit could be “wafer”. If it was a particular size of Wafer the Unit could be “200mmWafer.” */

```
void setUnit (
     in string aUnit)
     raises (Global::FrameworkErrorSignal,
          InvalidUnitSignal);
```

/* Get the Unit for the MaterialTrackingLocation. */

```
string getUnit ( )
     raises (Global::FrameworkErrorSignal);
```

/* Add a Material to the MaterialTrackingLocation. */

```
void addMaterialToTrack (
     in AbstractIF::Material aMaterial )
     raises (Global::FrameworkErrorSignal,
          InvalidMaterialSignal,
          MaterialTrackingLocationFullSignal);
```

/* Remove a Material from the MaterialTrackingLocation. */

```
void removeMaterial (
     in AbstractIF::Material aMaterial )
     raises (Global::FrameworkErrorSignal,
          InvalidMaterialSignal);
```

/* Reserve capacity of the MaterialTrackingLocation. If accepted, this portion of the MaterialTrackingLocation’s capacity will be kept for the designated material. If the capacity is reserved the operation returns True. */

```
boolean reserveCapacity (
     in AbstractIF::Material capacityToReserve)
     raises (InvalidUnitSignal,
          InvalidMaterialSignal,
          MaterialTrackingLocationFullSignal,
          Global::FrameworkErrorSignal);
```

/* Release reserved capacity of the MaterialTrackingLocation. If accepted, this portion of the MaterialTrackingLocation’s capacity will be released for use by other material. */

```
boolean releaseCapacity (
     in AbstractIF::Material capacityToRelease)
     raises (Global::FrameworkErrorSignal);
```

/* Get and set the Maximum Capacity for the MaterialTrackingLocation. */

```
void setMaximumCapacity (
     in long maximum)
     raises (Global::FrameworkErrorSignal,
          Global::SetValueOutOfRangeSignal);

long getMaximumCapacity ( )
     raises (Global::FrameworkErrorSignal);
```

/* Get the Available Capacity for the MaterialTrackingLocation. Available Capacity is the difference between Maximum Capacity and the count of the sequence returned by AllMaterialHeld. */

```
long getAvailableCapacity ( )
     raises (Global::FrameworkErrorSignal);
```

/* Get the Reserved Capacity for the MaterialTrackingLocation. Reserved Capacity is the portion of the AvailableCapacity that has been set aside for known material that has not yet arrived. */

```
long getReservedCapacity ( )
     raises (Global::FrameworkErrorSignal);
```

/* This is a query for the availability status of the MaterialTrackingLocation. Available means the MaterialTrackingLocation has available capacity in which more material could go. */

```
boolean hasAvailableCapacity ( )
     raises (Global::FrameworkErrorSignal);

}; // MaterialTrackingLocation
```

6.2.4.3 *Port Interface (subset)*

Module: MachineAIG

Interface: Port

Inherited Interface: Resource

```
interface Port : AbstractIF::Resource {
```

Description: The Port represents the point at which a “change of ownership” occurs during a material transfer. Each Port has at least one associated Machine and the Port may be thought of as an “access point” to the Machine.

Exceptions: None.

Events: None.

Provided Services:

/* Returns the machine to which this port belongs. */

```
Machine getMachine ( )
     raises (Global::FrameworkErrorSignal);

}; // Port
```

6.2.5 *Factory Operations Component (subset)* — This section includes only those interfaces and specific operations from the Factory Component that are referenced within this component specification. These interfaces are provided here for referential integrity of this specification and are not intended to remain as part of this component. Further, this is not intended to be a comprehensive treatment of the required interfaces, but only the subset needed for reference until the full specification is adopted through a subsequent ballot for the full Factory Operations component. When that occurs, this specification can be updated to reference the interfaces directly rather than replicate them here.

6.2.5.1 *MachineRegister Interface*

Module: FactoryOperations

Interface: MachineRegister

Inherited Interface: Resource

```
interface MachineRegister : AbstractIF::Resource {
```

Description: The MachineRegister maintains a list of known machines for the factory and supplies related information on demand. There is only one MachineRegister per factory. NOTE: this interface may eventually be replaced by the trader service.

Exceptions:

/* An attempt was made to locate an unknown Machine. */

```
exception MachineNotFoundSignal {string machineName;};
```

```
exception MachineRemovalFailedSignal { };
```

Published Events:

/* Event indicating that a Machine has been added or removed from the Factory. */

```
MachineListChangedEvent
```

Provided Services:

/* Add a Machine to the set of machine(s) managed by the MachineRegister. */

```
void addMachine (in Machine aMachine)
     raises (Global::FrameworkErrorSignal);
```

/* Remove a Machine from the set of machine(s) managed by the MachineRegister. */

```
void removeMachine (in Machine aMachine)
     raises (Global::FrameworkErrorSignal,
          MachineRemovalFailedSignal);
```

/* Return a sequence of all machines managed by the MachineRegister. */

```
MachineSequence allMachines ( )
     raises (Global::FrameworkErrorSignal);
```

/* Return the Machine corresponding to the given name. */

```
Machine findMachineNamed (in string identifier)
     raises (Global::FrameworkErrorSignal,
          MachineNotFoundSignal) ;
```

Contracted Services: None.

Dynamic Model: None.

```
}; // MachineRegister
```

6.3 *Material Transport Abstract Interfaces* — The Material Transport Abstract Interfaces are defined to perform material transport at the factory level. The TransportJobSupervisor interface must be specialized by the users of the Material Transport Manager Interface. These interfaces provide a way to track the job through the factory.

6.3.1 *TransportJobSupervisor Interface*

Module: MaterialTransport

Interface: TransportJobSupervisor

Inherited Interface: JobSupervisor

```
interface TransportJobSupervisor : AbstractIF::JobSupervisor {
```

Description: TransportJobSupervisor is an abstract interface which provides the operations needed by clients to request, track, and control material transport within the domain of the specific TransportJobSupervisor. Any entity which is responsible for moving material will inherit and implement TransportJobSupervisor.

Type Definitions:

```
typedef sequence <Global::Properties> JobSpecSequence;
```

Exceptions:

```
exception UnsupportedDestinationTypeSignal { };
```

Published Events: None.

Provided Services:

The Creation of TransportJobs is accomplished by use of the requestJob operation from the inherited JobSupervisor interface. The "JobType" property specified in the inherited interface should be set to "TransportJobType" for all TransportJob requests. The additional Job Specification properties required for the creation of Transport jobs are defined in Table 3. The types of machines specified in the TransportJob are AMHS Storage Machines, Fixed Buffer Production Equipment, or Internal Buffer Production Equipment. Each machine type represents a separate case used in defining the Job Specification for a Transport Job, and these cases are also defined in Table 3.

For transportJobs, the inherited ABORTING and ABORTED states may not always result in an immediate termination of the job and associated activities. For example, transportJobs may involve autonomous transport equipment that operate without persistent communication links to the TransportJob and TransportJobSupervisor objects. In these cases there may be a time delay before the results of a transportJob makeAborted request can be enacted to allow the transport equipment to reach a known location. In other cases a movement activity must reach a stable condition before it can be interrupted. The location of the MaterialContainer associated with the aborted TransportJob is the nearest location where movement can be safely and reliably interrupted.

By contrast, if a transportJob is transitioned to the inherited STOPPING state, the location of the MaterialContainer associated with the stopped TransportJob must be a legal Source for issuing a new TransportJob.

Table 3 Additional Required Properties for Transport Jobs

Name	*Value Type*	*Description*
"Container"	MaterialContainer	Container being transported.
"SourceMachine"	Machine	The source production machine for the transport job. AMHS Storage Equipment: SourceMachine is Optional. Fixed Buffer Production Equipment: SourceMachine is Required. Internal Buffer Production Equipment: SourceMachine is Required.
"SourcePort"	Port	The source production machine port for the transport job. AMHS Storage Equipment: SourcePort is Optional. Fixed Buffer Production Equipment: SourcePort is Required. Internal Buffer Production Equipment: SourcePort is optional. Port may be obtained dynamically with the containerOut operation or it may be predefined in the TransportJob specification.
"DestinationArea"	Area	The destination area for the transport job. DestinationArea shall contain only StorageMachines. The implementation shall determine the specific machine to which the MaterialContainer is delivered. AMHS Storage Equipment: DestinationArea is optional. Fixed Buffer Production Equipment: DestinationArea is not allowed. Internal Buffer Production Equipment: DestinationArea is not allowed.
"DestinationMachine"	Machine	The destination machine for the transport job. AMHS Storage Equipment: DestinationMachine is required if DestinationArea is not specified. Fixed Buffer Production Equipment: DestinationMachine is required. Internal Buffer Production Equipment: DestinationMachine is required.

Name	*Value Type*	*Description*
"DestinationPort"	Port	The destination port for the transport job. AMHS Storage Equipment: DestinationPort is required for Manual output port destinations. Fixed Buffer Production Equipment: DestinationPort is required. Internal Buffer Production Equipment: DestinationPort is NULL.

/* The requestBatchTransportJob operation will provide two or more job specifications for transport jobs that share a dependency on completion. All job specifications must have the same priority. In the normal case, the TransportJobSupervisor will create simple transport jobs for each job specification and return the jobs as a sequence. Subsequently, if one of the simple jobs cannot complete, the other subordinate jobs must be reassessed by the JobRequestor to determine the appropriate corrective action. In this case, the TransportJobSupervisor will notify the JobRequestor of the job failure with the informJobTerminated message and pause the other jobs of the batch-job to allow the JobRequestor to determine its response. Implementations may implement the TransportJobSupervisor with more complex default actions in response to a partial failure of a batch-transport-job, but these more complex scenarios are not part of the standard behavior for this operation. */

```
AbstractIF::JobSequence requestBatchTransportJob (
     in JobSpecSequence jobSpecs,
     in AbstractIF::JobRequestor aJobRequestor)
     raises (Global::FrameworkErrorSignal,
          AbstractIF::JobSupervisor::JobRejectedSignal);
```

Contracted Services: None

Dynamic Model: None

```
}; // TransportJobSupervisor
```

6.4 *Material Transport and Storage Component*

6.4.1 This component is concerned only with factory-wide material transport and not the movement of material within a piece of equipment. Also note that material can be either product, durables, or consumables. The CIM Framework recognizes the potential for multiple AMHS's within a single factory (e.g., one supplier of interbay movement and another supplier of intrabay movement). The MES, however, wants to only have to issue a move request ("move this material from here to there") without regard to whether that transport spans one or more AMHS's. Therefore, the component supporting material transport requires a single entry point for initiating material transport which will be through the Material Transport Manager (MTM).

6.4.2 The Material Transport Manager must be able manage transport jobs carried out by many different AMHS's. To accomplish this, a separate interface for a Material Transport Controller (MTC) is specified. In the simplest implementations, a single MTM may manage all TransportJobs directly. In this case no separate Material Transport Controller implementation is needed. If there is a need for delegation to one or more Material Transport Controllers, the Material Transport Manager can register any conformant MTC implementation which supports the specified interface. The low-level interaction of the material movement controllers with the physical equipment (transport machines, storage machines and ports) is encapsulated within the scope of this component. Note that the component manager does not provide lifecycle services for its managed material movement controllers, but instead uses the registration interfaces.

6.4.3 *MaterialTransportManager Interface*

Module: MaterialTransport

Interface: MaterialTransportManager

Inherited Interface: ComponentManager, TransportJobSupervisor, JobRequestor

```
interface MaterialTransportManager : FactoryOperations::ComponentManager,
     TransportJobSupervisor, AbstractIF::JobRequestor {
```

Description: A MaterialTransportManager (MTM) operates on requests to move material in the factory. It returns a TransportJob to the requestor, allowing the requestor to track the status and progress of the move. The MTM may perform these TransportJobs by breaking them into separate tasks to be performed by MaterialTransportControllers (MTC). If TransportJobs are decomposed, the MaterialTransportManager then tracks the MTC TransportJobs to fulfill its commitment.

Exceptions:

```
exception MaterialTransportControllerNotRegisteredSignal { };
```

Published Events: None.

Provided Services:

/* Add a MaterialTransportController to the collection of MaterialTransportControllers which the MaterialTransportManager can use to carry out jobs. */

```
void registerMaterialTransportController (
     in MaterialTransportController aMaterialTransportController)
     raises (Global::FrameworkErrorSignal);
```

/* Remove a MaterialTransportController from the list of resources available to the MaterialTransportManager. No new jobs will be issued but existing jobs will be unaffected. */

```
void unregisterMaterialTransportController (
     in MaterialTransportController aMaterialTransportController)
     raises (Global::FrameworkErrorSignal,
          MaterialTransportControllerNotRegisteredSignal);
```

/* Return a list of all MaterialTransportControllers managed by the MaterialTransportManager. */

```
MaterialTransportControllerSequence allMaterialTransportControllers ( )
     raises (Global::FrameworkErrorSignal);
```

/* Returns a list of all material in the material transport system. */

```
AbstractIF::MaterialSequence allMaterial ( )
     raises (Global::FrameworkErrorSignal);
```

/* Returns a list of all material in transit in the material transport system. */

```
AbstractIF::MaterialSequence allMaterialInTransit ( )
     raises (Global::FrameworkErrorSignal);
```

/* Returns a list of all material in storage in the material transport system. */

```
AbstractIF::MaterialSequence allMaterialInStorage ( )
     raises (Global::FrameworkErrorSignal);
```

Contracted Services: None

Dynamic Model: None

```
}; // MaterialTransportManager
```

6.4.4 *MaterialTransportController Interface*

Module: MaterialTransport

Interface: MaterialTransportController

Inherited Interface: TransportJobSupervisor

```
interface MaterialTransportController : TransportJobSupervisor {
```

Description: A MaterialTransportController accepts transport job requests, schedules them, and executes them in order to move material around the factory or a part of the factory.

Exceptions: None.

Published Events: None.

Provided Services:

```
typedef sequence <MaterialTransportController> MaterialTransportControllerSequence;
```

/* Returns a list of all material in the domain of this material transport controller. */

```
AbstractIF::MaterialSequence allMaterial ( )
     raises (Global::FrameworkErrorSignal);
```

/* Returns a list of all material in transit in the domain of this material transport controller. */

```
AbstractIF::MaterialSequence allMaterialInTransit ( )
     raises (Global::FrameworkErrorSignal);
```

/* Returns a list of all material in storage in the domain of this material transport controller. */

```
AbstractIF::MaterialSequence allMaterialInStorage ( )
     raises (Global::FrameworkErrorSignal);
```

Contracted Services: None

Dynamic Model: None

```
}; // MaterialTransportController
```

6.4.5 *TransportJob Interface*

Module: MaterialTransport

Interface: TransportJob

Inherited Interface: Job

```
interface TransportJob : AbstractIF::Job {
```

Description: This type of Job performs a specific transport of material. The ongoing status of the move can be monitored by subscribing to the appropriate Job events.

Exceptions:

```
exception TimeUndeterminableSignal { };
```

Published Events: Same as Job

Provided Services:

/* Determine if the move can be completed by the specified time. */

```
boolean canCompleteBy (
     in Global::TimeStamp whenNeeded)
     raises (Global::FrameworkErrorSignal,
          TimeUndeterminableSignal);
```

/* Estimate how long this job will take once it begins. */

```
Global::Duration transportTime ( )
     raises (Global::FrameworkErrorSignal,
          TimeUndeterminableSignal);
```

Contracted Services: None

Dynamic Model: None

```
}; // TransportJob
```

6.4.6 *TransportMachine Interface*

Module: MaterialTransport

Interface: TransportMachine

Inherited Interface: Machine

```
interface TransportMachine : MachineAIG::Machine {
```

Description: This concrete specialization of Machine represents the equipment used to transport material within the factory. Implementations of the Material Transport and Storage Component would provide implementations supporting this interface for each TransportMachine registered with the factory.

Exceptions: None

Published Events: Same as Machine

Provided Services: None

Contracted Services: None

Dynamic Model: None

```
}; // TransportMachine
```

6.4.7 *StorageMachine Interface*

Module: MaterialTransport

Interface: StorageMachine

Inherited Interface: Machine

```
interface StorageMachine : MachineAIG::Machine {
```

Description: This concrete specialization of Machine represents the equipment used to store material within the factory. Implementations of the Material Transport and Storage Component would provide implementations supporting this interface for each StorageMachine registered with the factory.

Exceptions: None

Published Events:

*/ Notifies subscribers that a MaterialContainer arrived on a manual input port may require further action (e.g., creation of a TransportJob or transfer to a particular logical partition). */

```
const string MaterialContainerArrivedAtManualInputPortSubject =
     "/MaterialTransport/StorageMachine/MaterialContainerArrivedAtManualInputPort";

struct MaterialContainerArrivedAtManualInputPortFilters {
     Global::Property Port;
     Global::Property MaterialContainer;
};
```

Table 4 MaterialContainerArrivedAtManualInputPortFilters Properties:

Name	*Value Type*	*Description*
"Port"	MachineAIG::Port	The port where the MaterialContainer was placed.
"MaterialContainer"	DurablesManagement::Material Container	The carrier that arrived at the manual input port.

```
struct MaterialContainerArrivedAtManualInputPortEvent {
     string eventSubject;
     Global::TimeStamp eventTimeStamp;
     MaterialContainerArrivedAtManualInputPortFilters eventFilterData;
     Global::Properties eventNews;
     StorageMachine aMachine
};
```

Provided Services: None

Contracted Services: None

Dynamic Model: None

```
}; // StorageMachine
```

APPENDIX 1
COMPLETE LISTING OF MATERIAL TRANSPORT IDL

NOTE: The material in this appendix is an official part of SEMI E10# and was approved by full letter ballot procedures on January 14, 2000 by the Japanese Regional Standards Committee.

```
module CIMFW {

#include <Global.idl>
#include <FactoryLabor.idl>
#include <AbstractIF.idl>
#include <FactoryOperations.idl>

module MachineAIG {

     interface Machine;

     interface MaterialTrackingLocation;

     interface Port;

     typedef sequence <Machine> MachineSequence;

     exception MachineDuplicateSignal { };

     exception MachineNotAssignedSignal { };

     exception MachineRemovalFailedSignal { };

     exception MaterialTrackingLocationFullSignal { };

     interface Machine : AbstractIF::Resource {

          typedef sequence <Machine> MachineSequence;

          typedef sequence <MaterialTrackingLocation>
          MaterialTrackingLocationSequence;

          typedef sequence <Port> PortSequence;

          enum MachineType {
               FixedBufferProductionMachine,
               InternalBufferProductionMachine,
               StorageMachine,
               TransportMachine };

          exception InvalidPortSignal { };

          exception InvalidMaterialSignal { };

          void setMachineType (
               in MachineType aMachineType )
               raises (Global::FrameworkErrorSignal);

          MachineType getMachineType ( )
               raises (Global::FrameworkErrorSignal);
```

```
        void setMaterialTrackingLocation (
              in MaterialTrackingLocation aMaterialTrackingLocation,
              in string aUnitToTrack)
              raises (Global::FrameworkErrorSignal);

        MaterialTrackingLocationSequence getAllMaterialTrackingLocations ( )
              raises (Global::FrameworkErrorSignal);

        MaterialTrackingLocation getMaterialTrackingLocation (
              in string aUnit)
              raises (Global::FrameworkErrorSignal);

        Port reservePortForTransferTo (
              in DurablesManagement::MaterialContainer aContainer,
              in Port requestedPort )
              raises (Global::FrameworkErrorSignal,
                    InvalidPortSignal);

        Port reservePortForBatchTransferTo (
              in DurablesManagement::MaterialContainerSequence aContainerGroup,
              in Port requestedPort )
              raises (Global::FrameworkErrorSignal,
                    InvalidPortSignal);

        void cancelPortReservationForTransferTo (
              in DurablesManagement::MaterialContainer aContainer,
              in Port requestedPort )
              raises (Global::FrameworkErrorSignal,
                    InvalidPortSignal);

        void transferComplete (in DurablesManagement::MaterialContainer aContainer,
              in Port reservedPort )
              raises (Global::FrameworkErrorSignal,
                    InvalidPortSignal);

        Port containerOut (
              in DurablesManagement::MaterialContainer aContainer,
              in Port requestedPort )
              raises (Global::FrameworkErrorSignal,
                    InvalidMaterialSignal);

        PortSequence allPorts ( )
              raises (Global::FrameworkErrorSignal);

        FactoryOperations::AreaSequence allAreas( )
              raises (Global::FrameworkErrorSignal);

  }; // Machine

  interface MaterialTrackingLocation : AbstractIF::OwnedEntity {

        exception InvalidMaterialSignal { };

        exception MaterialTrackingLocationFullSignal { };

        exception InvalidUnitSignal { };

        AbstractIF::MaterialSequence allMaterialHeld ( )
              raises (Global::FrameworkErrorSignal);
```

```
        string getIdentifier ( )
              raises (Global::FrameworkErrorSignal);

        void setIdentifier (
              in string identifier)
              raises (Global::FrameworkErrorSignal,
                    Global::SetValueOutOfRangeSignal,
                    Global::DuplicateIdentifierSignal);

        Machine getMachine ( )
              raises (Global::FrameworkErrorSignal);

        void setUnit (
              in string aUnit)
              raises(Global::FrameworkErrorSignal,
              InvalidUnitSignal);

        string getUnit ( )
              raises (Global::FrameworkErrorSignal);

        void addMaterialToTrack (
              in AbstractIF::Material aMaterial )
              raises (Global::FrameworkErrorSignal,
                    InvalidMaterialSignal,
                    MaterialTrackingLocationFullSignal);

        void removeMaterial (
              in AbstractIF::Material aMaterial )
              raises (Global::FrameworkErrorSignal,
                    InvalidMaterialSignal);

        boolean reserveCapacity (
              in AbstractIF::Material capacityToReserve)
              raises (InvalidUnitSignal,
                    InvalidMaterialSignal,
                    MachineAIG::MaterialTrackingLocationFullSignal,
                    Global::FrameworkErrorSignal);

        boolean releaseCapacity (
              in AbstractIF::Material capacityToRelease)
              raises (Global::FrameworkErrorSignal);

        void setMaximumCapacity (
              in long maximum)
              raises (Global::FrameworkErrorSignal,
                    Global::SetValueOutOfRangeSignal);

        long getMaximumCapacity ( )
              raises (Global::FrameworkErrorSignal);

        long getAvailableCapacity ( )
              raises (Global::FrameworkErrorSignal);

        long getReservedCapacity ( )
              raises (Global::FrameworkErrorSignal);

        boolean hasAvailableCapacity ( )
              raises (Global::FrameworkErrorSignal);

  }; // MaterialTrackingLocation
```

```
     interface Port : AbstractIF::Resource {

          MachineAIG::Machine getMachine ( )
               raises (Global::FrameworkErrorSignal);

     }; // Port

}; // module MachineAIG

module DurablesManagement {

     interface Durable : AbstractIF::Material {

          void setMaterialTrackingLocation (
               in MachineAIG::MaterialTrackingLocation aMaterialTrackingLocation)
               raises (Global::FrameworkErrorSignal,
                    MachineAIG::
          MaterialTrackingLocation::MaterialTrackingLocationFullSignal);

          MachineAIG::MaterialTrackingLocation getMaterialTrackingLocation ( )
               raises (Global::FrameworkErrorSignal);

          void setUnit(
               in string aUnit)
               raises (Global::FrameworkErrorSignal,
                    MachineAIG::MaterialTrackingLocation::InvalidUnitSignal);

          string getUnit( )
               raises (Global::FrameworkErrorSignal);

     }; // Durable

     interface MaterialContainer : Durable {

          void makeManualControl ( )
               raises (Global::FrameworkErrorSignal,
                    Global::InvalidStateTransitionSignal);

          void makeStored ( )
               raises (Global::FrameworkErrorSignal,
                    Global::InvalidStateTransitionSignal);

          void makeInTransit ( )
               raises (Global::FrameworkErrorSignal,
                    Global::InvalidStateTransitionSignal);

          void makeProcessing ( )
               raises (Global::FrameworkErrorSignal,
                    Global::InvalidStateTransitionSignal);

     }; // MaterialContainer

     typedef sequence <MaterialContainer> MaterialContainerSequence;

}; // module DurablesManagement {

#ifndef _CIMFW_MATERIAL_TRANSPORT_

#define _CIMFW_MATERIAL_TRANSPORT_
```

```
module MaterialTransport {

     interface MaterialTransportController;

     typedef sequence <MaterialTransportController>
     MaterialTransportControllerSequence;

     interface TransportJobSupervisor : AbstractIF::JobSupervisor {

          typedef sequence <Global::Properties> JobSpecSequence;

          exception UnsupportedDestinationTypeSignal { };

          AbstractIF::JobSequence requestBatchTransportJob (
               in JobSpecSequence jobSpecs,
               in AbstractIF::JobRequestor aJobRequestor)
               raises (Global::FrameworkErrorSignal,
                    AbstractIF::JobSupervisor::JobRejectedSignal);

     }; // TransportJobSupervisor

     interface MaterialTransportManager : FactoryOperations::ComponentManager,
          TransportJobSupervisor, AbstractIF::JobRequestor {

          exception MaterialTransportControllerNotRegisteredSignal { };

          void registerMaterialTransportController (
               in MaterialTransportController aMaterialTransportController)
               raises (Global::FrameworkErrorSignal);

          void unregisterMaterialTransportController (
               in MaterialTransportController aMaterialTransportController)
               raises (Global::FrameworkErrorSignal,
                    MaterialTransportControllerNotRegisteredSignal);

          MaterialTransportControllerSequence allMaterialTransportControllers ( )
               raises (Global::FrameworkErrorSignal);

          AbstractIF::MaterialSequence allMaterial ( )
               raises (Global::FrameworkErrorSignal);

          AbstractIF::MaterialSequence allMaterialInTransit ( )
               raises (Global::FrameworkErrorSignal);

          AbstractIF::MaterialSequence allMaterialInStorage ( )
               raises (Global::FrameworkErrorSignal);

     }; // MaterialTransportManager

     interface MaterialTransportController : TransportJobSupervisor {

          AbstractIF::MaterialSequence allMaterial ( )
               raises (Global::FrameworkErrorSignal);

          AbstractIF::MaterialSequence allMaterialInTransit ( )
               raises (Global::FrameworkErrorSignal);

          AbstractIF::MaterialSequence allMaterialInStorage ( )
               raises (Global::FrameworkErrorSignal);
```

```
    }; // MaterialTransportController

    interface TransportJob : AbstractIF::Job {

        exception TimeUndeterminableSignal { };

        boolean canCompleteBy (
            in Global::TimeStamp whenNeeded)
            raises (Global::FrameworkErrorSignal,
                TimeUndeterminableSignal);

        Global::Duration transportTime ( )
            raises (Global::FrameworkErrorSignal,
                TimeUndeterminableSignal);

    }; // TransportJob

    interface TransportMachine : MachineAIG::Machine {

    }; // TransportMachine

    interface StorageMachine : MachineAIG::Machine {

        const string MaterialContainerArrivedAtManualInputPortSubject =
            "/MaterialTransport/StorageMachine/
            MaterialContainerArrivedAtManualInputPort";

        struct MaterialContainerArrivedAtManualInputPortFilters {
            Global::Property Port;
            Global::Property MaterialContainer;
        };

        struct MaterialContainerArrivedAtManualInputPortEvent {
            string eventSubject;
            Global::TimeStamp eventTimeStamp;
            MaterialContainerArrivedAtManualInputPortFilters eventFilterData;
            Global::Properties eventNews;
            StorageMachine aMachine
        };

    }; // StorageMachine

}; // module MaterialTransport

#endif // _CIMFW_MATERIAL_TRANSPORT_

}; // module CIMFW
```

RELATED INFORMATION 1
SCENARIOS FOR MATERIAL TRANSPORT AND STORAGE

NOTE: This related information is not an official part of SEMI E10# and was derived from the work of the I300I/J300E AMHS workgroup accomplished during development of the proposed standard. This related information is included with the Material Transport and Storage Component specification to aid the readers in understanding the intent and use of the standard. This related information was approved for publication by full letter ballot procedures on January 14, 2000.

R1-1 Introduction

R1-1.1 *Scenario Assumptions* — These scenarios show possible implementations of the SEMI standard. The information shown here is intended to be a guide rather than describe rigid implementation rules. Some of the message sequences and implied functionality included in these scenarios will vary across different implementations.

R1-1.2 *Scenario Configuration* — There are thirteen scenarios included in this section. The first and last scenarios show all interactions at an individual interface level. For simplicity, the remaining scenarios show interactions abstracted up to the Component level. Background information related to the scenarios is shown in Figure R1-1. Table R1-1 outlines the thirteen scenario case conditions.

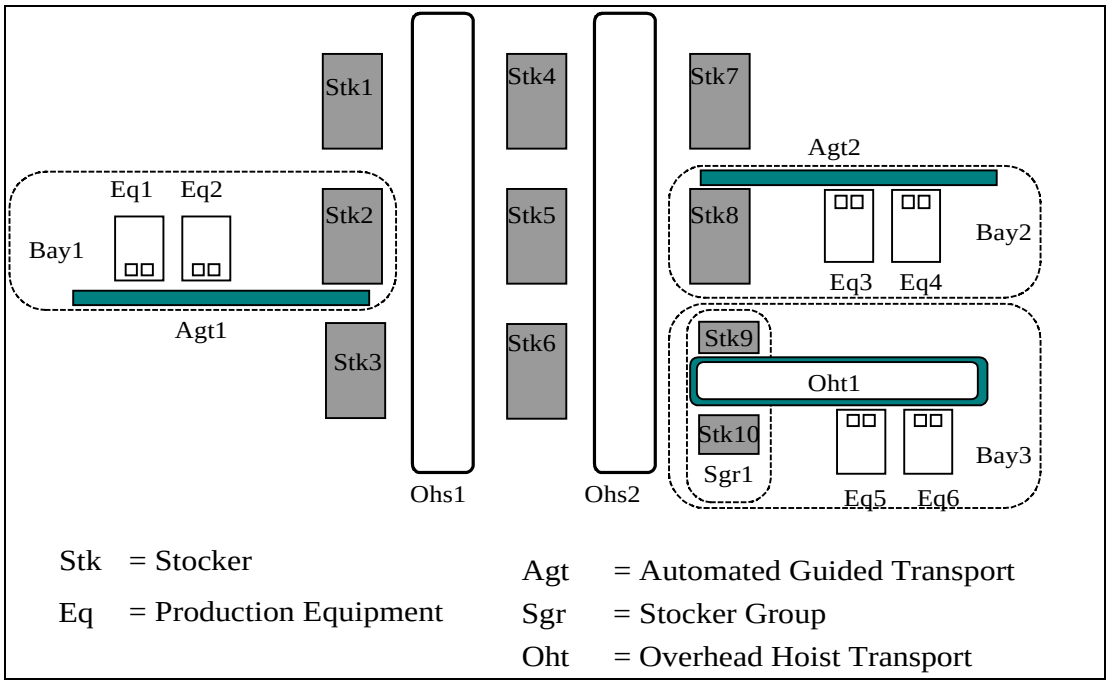

Figure R1-1
Scenario System Layout Configuration

Table R-1 Material Transport and Storage Scenario Definitions

Case	*Source*	*Destination*	*Batch Size*	*Route/Condition*	*Comments*
1	Stk4	Stk9 Manual Output Port	1 Carrier	Stk4>Ohs2>Stk9	Interactions at an individual interface level.
2	Eq1	Stk7	1 Carrier	Eq1>Agt1>Stk2>Ohs1>Stk(Relay)>Ohs2>Stk7 Stk(Relay) priority = Stk4, Stk5, Stk6	Base Scenario: Delivery from Production Equipment to Stocker.
3	Stk7	Eq3	2 Carriers	Stk7>Ohs2>Stk8>Agt2>Eq3 Eq3 batch size = 2 Carrier Ohs2 batch size = 1 Carrier Agt2 batch size = 2 Carrier	Base Scenario: Batch delivery from Stocker to Production Equipment.
4	Eq1	Stk7	1 Carrier	Eq1>Agt1>Stk2>Ohs1>Stk (Relay)>Ohs2>Stk8 (Temp)>Stk7 Stk(Relay) priority = Stk4, Stk5, Stk6 Stk4 Status = Down Stk7 logical partition = Full	Includes AMHS equipment (Storage Device) monitoring and stocker logical partition overflow control.
5	Stk6	Stk4	1 Carrier	Operator>Stk6	Manual Carrier Input handling.
6	Stk4	Stk9	1 Carrier	Stk4>Ohs2>Stk9 - Ohs2 Status = Down	Includes AMHS equipment (transport device) monitoring.
7	Stk7	Eq6	4 Carriers	Stk7>Ohs2>Sgr1>Oht1>Eq6 Eq6 has internal buffer, batch size = 4 carriers Ohs2 batch size = 1 carrier, Oht1 batch size = 1 carrier	Base scenario: Batch delivery to internal buffer equipment.
8	Stk5	Eq5	1 Carrier	Stk5>Ohs2>Sgr1>Oht1>Eq5 Macro command is aborted with carrier on Ohs2.	Includes abort job handling.
9	Stk5	Eq5	1 Carrier	Stk5>Ohs2>Sgr1>Oht1>Eq5 Macro command is modified to final destination Sgr1.	Includes modify job handling.
10	-	-	-	Startup - Material Transport and Storage Component is registered and starts up.	Base scenario: MTSC startup.
11	Stk6	Stk4	1 Carrier	Stk6>Ohs2>Stk4 - Operator removes carrier while on Ohs2.	Includes handling of manual interruption of delivery.
12	Eq3	Eq5	1 Carrier	Eq3>Agt2>Stk8>Ohs2>Sgr1>Oht1>Eq5	Base scenario: tool-to-tool delivery.
13	Stk4	Stk9	1 Carrier	Stk4>Ohs2>Stk9	Interactions at an individual interface level (including capacity reservation).

R1-2 Scenarios

R1-2.1 *Scenario One — Interactions at the Interface Level*

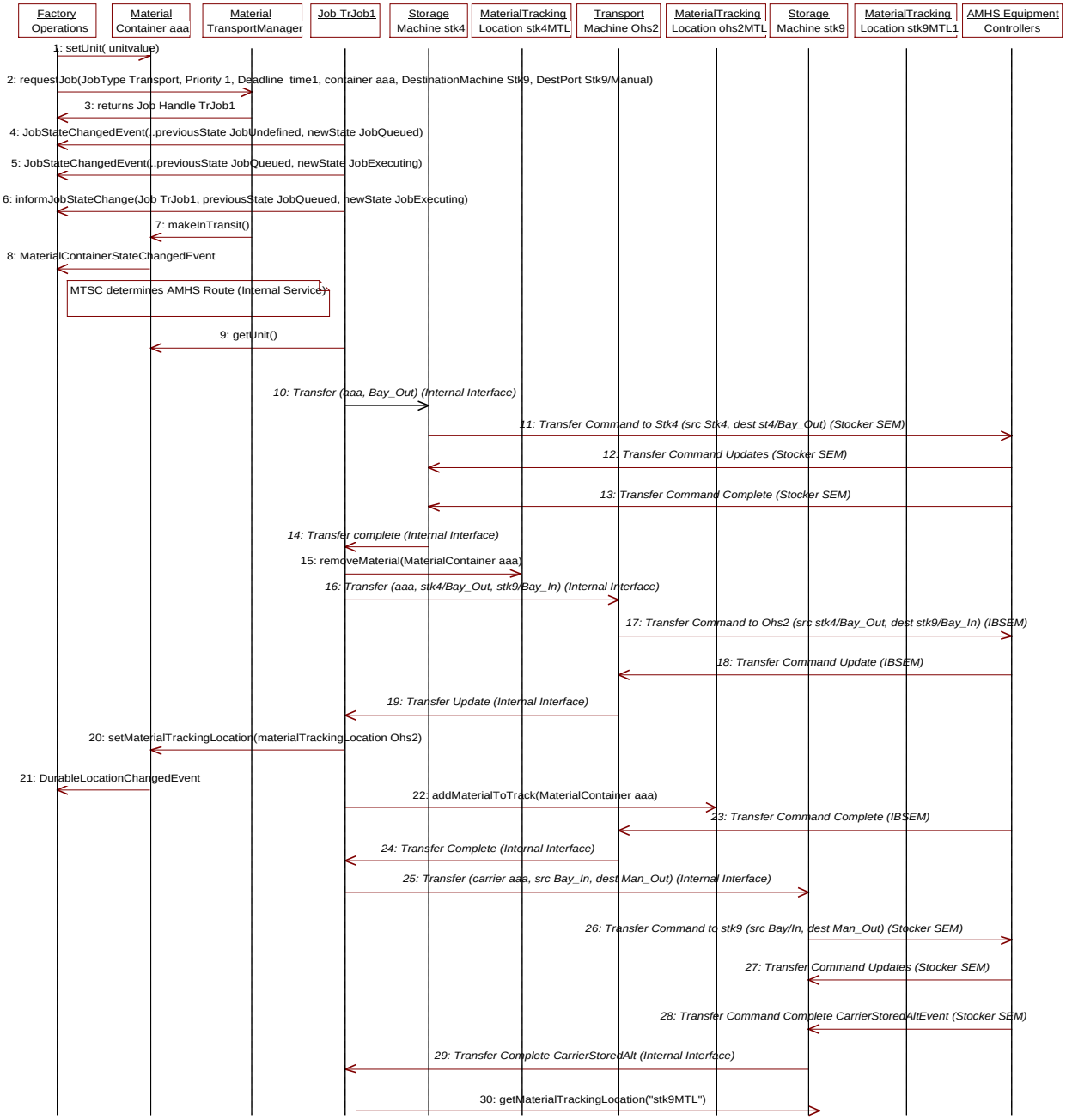

Figure R1-2
Scenario Case 1

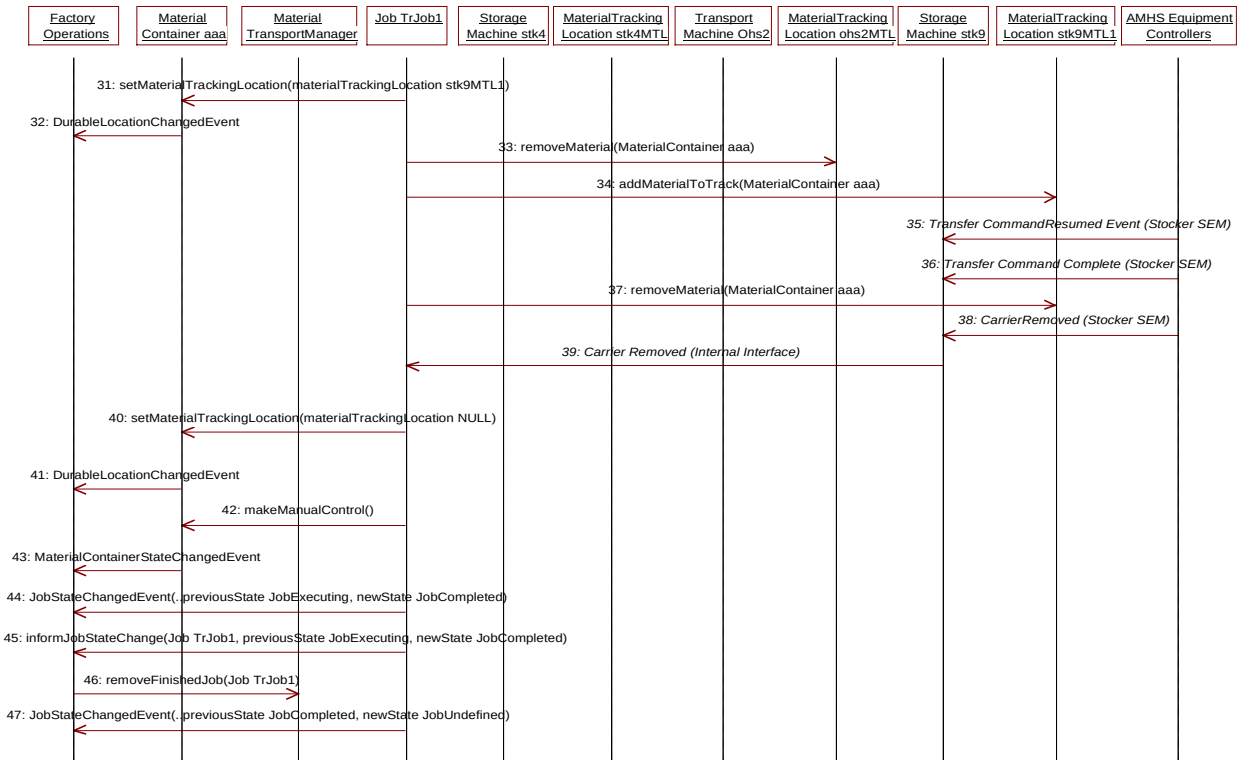

Figure R1-3
Scenario Case 1 (continued)

R1-2.2 *Scenario Two — Base Scenario for Delivery from Production Equipment to Stocker*

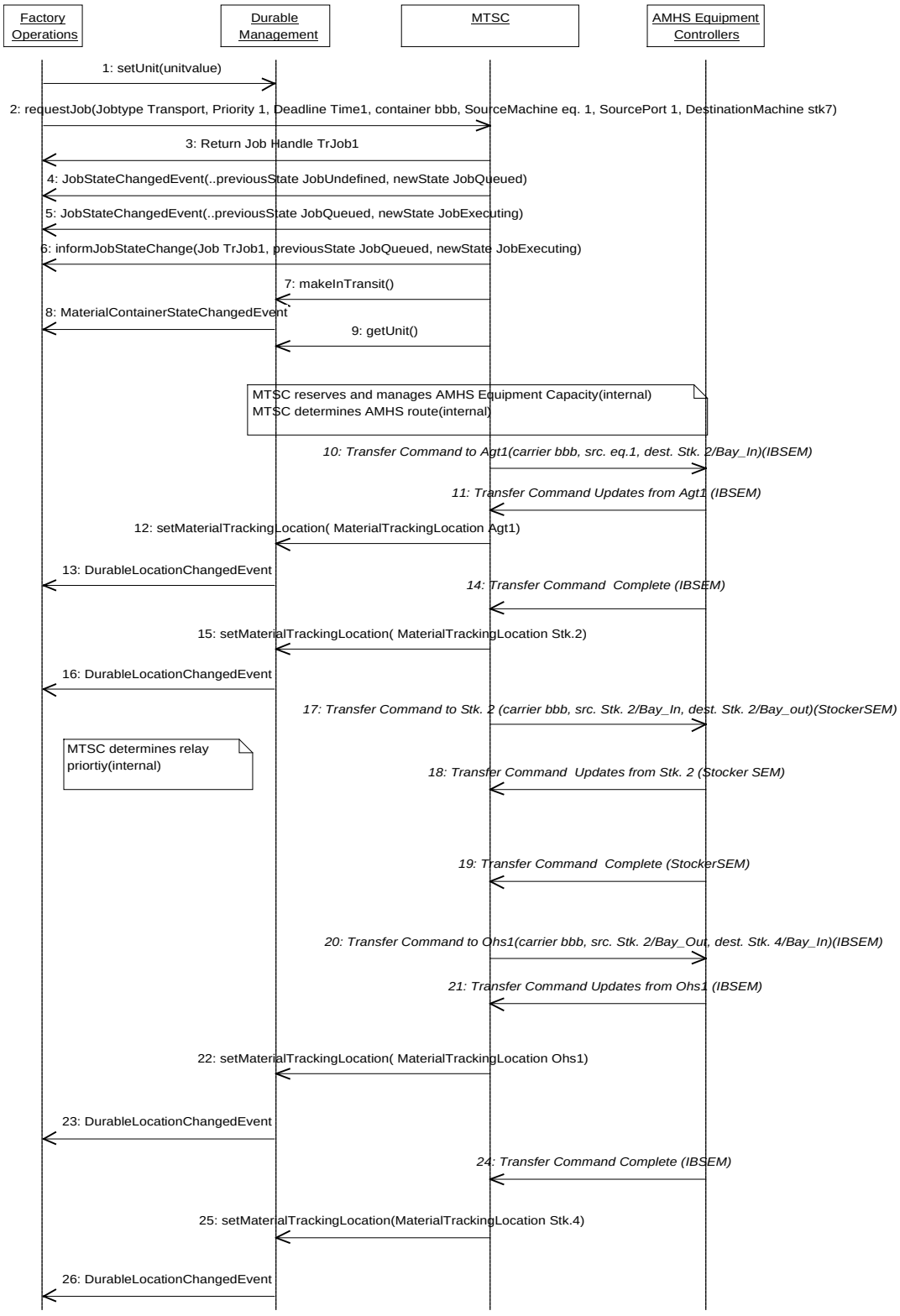

Figure R1-4
Scenario Case 2

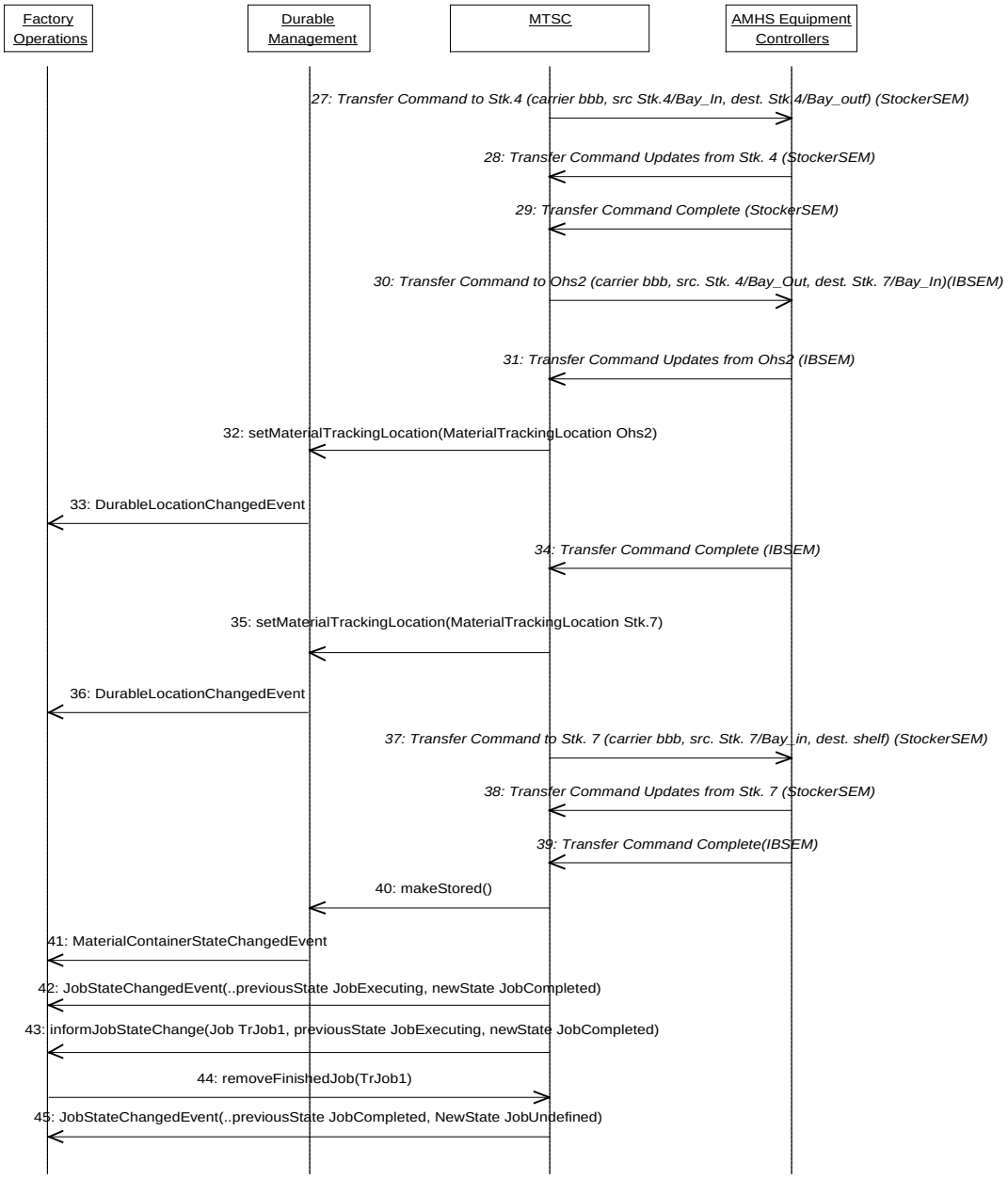

Figure R1-5
Scenario Case 2 (continued)

R1-2.3 *Scenario Three — Base Scenario for Batch Delivery from Stocker to Production Equipment*

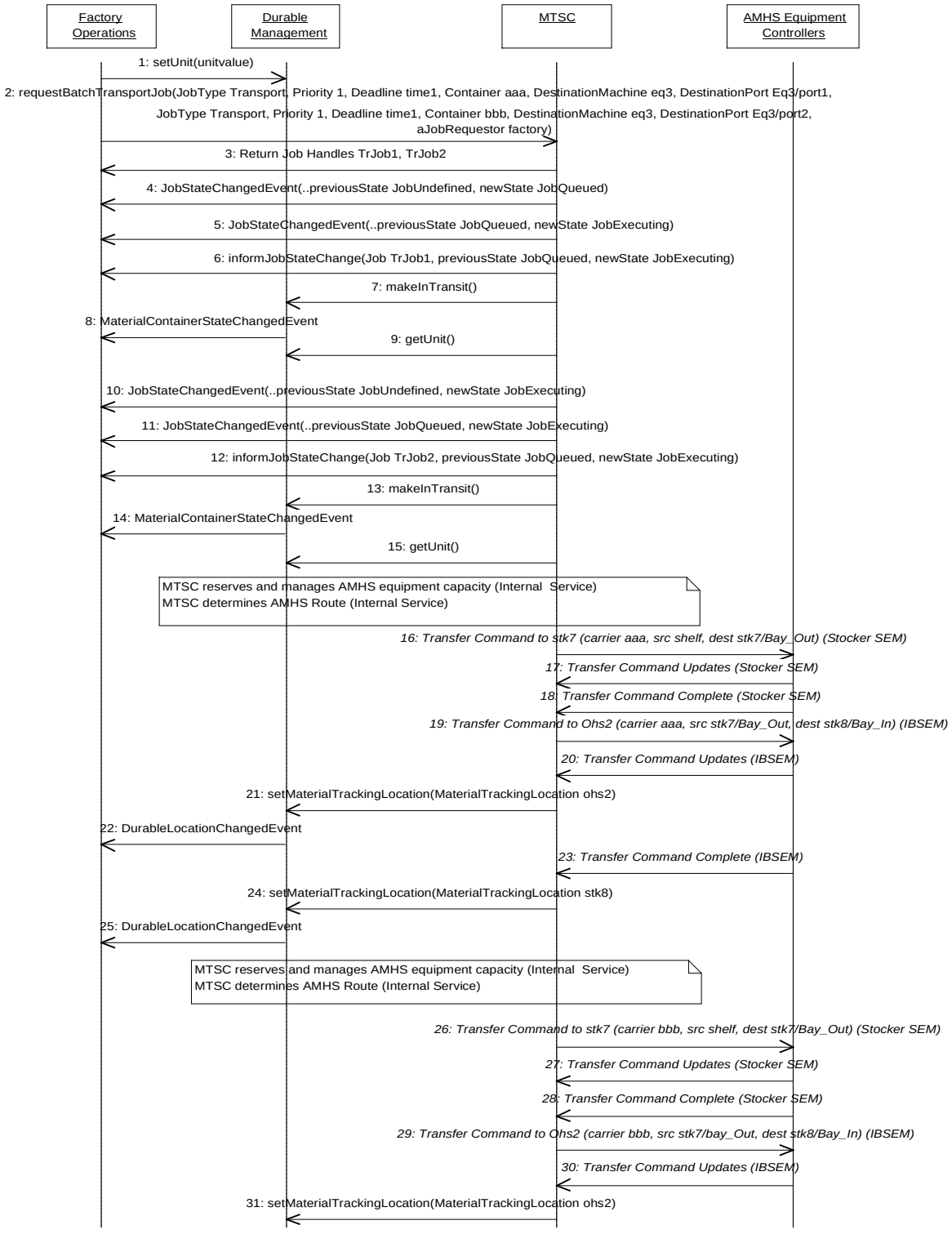

Figure R1-6
Scenario Case 3

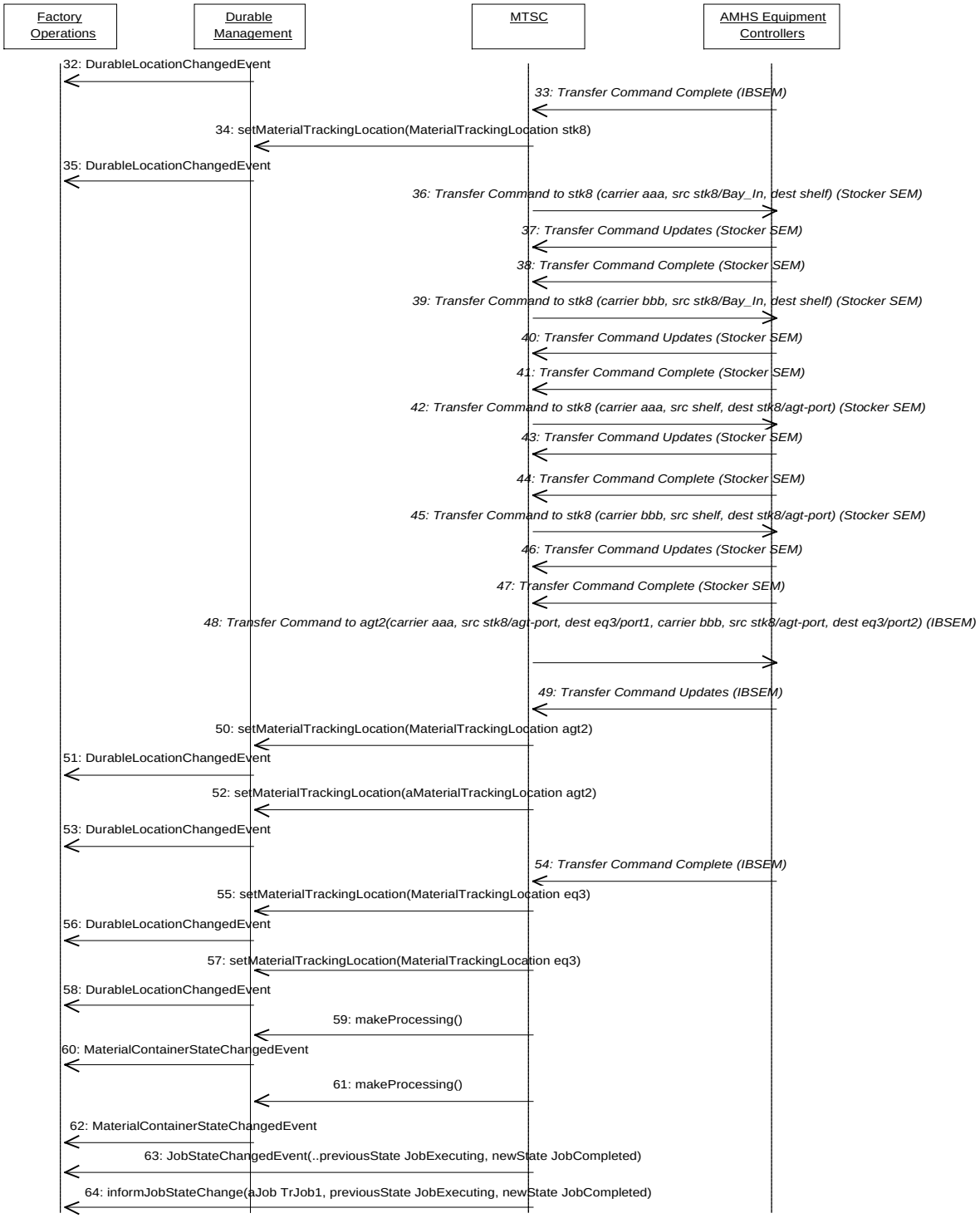

Figure R1-7
Scenario Case 3 (continued)

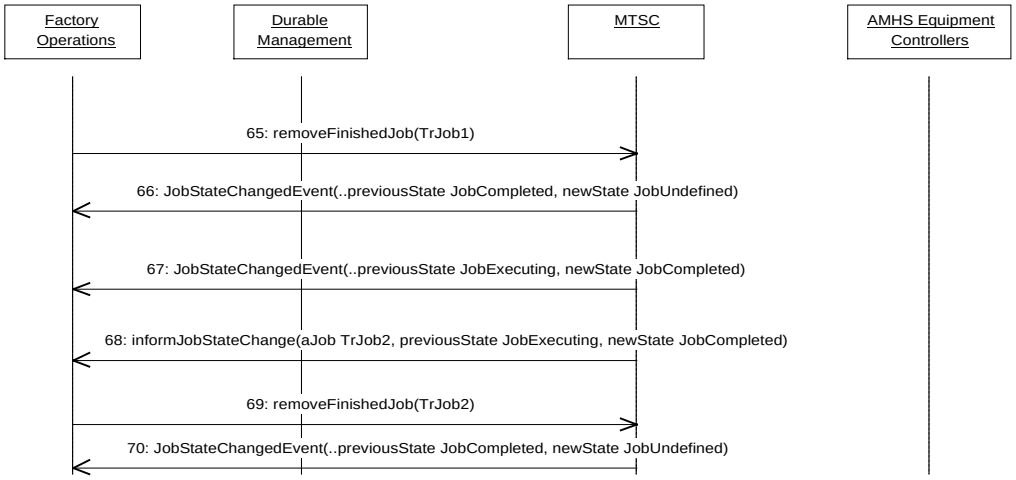

Figure R1-8
Scenario Case 3 (continued)

R1-2.4 *Scenario 4 — Equipment Monitoring and Logical Partition Overflow Control*

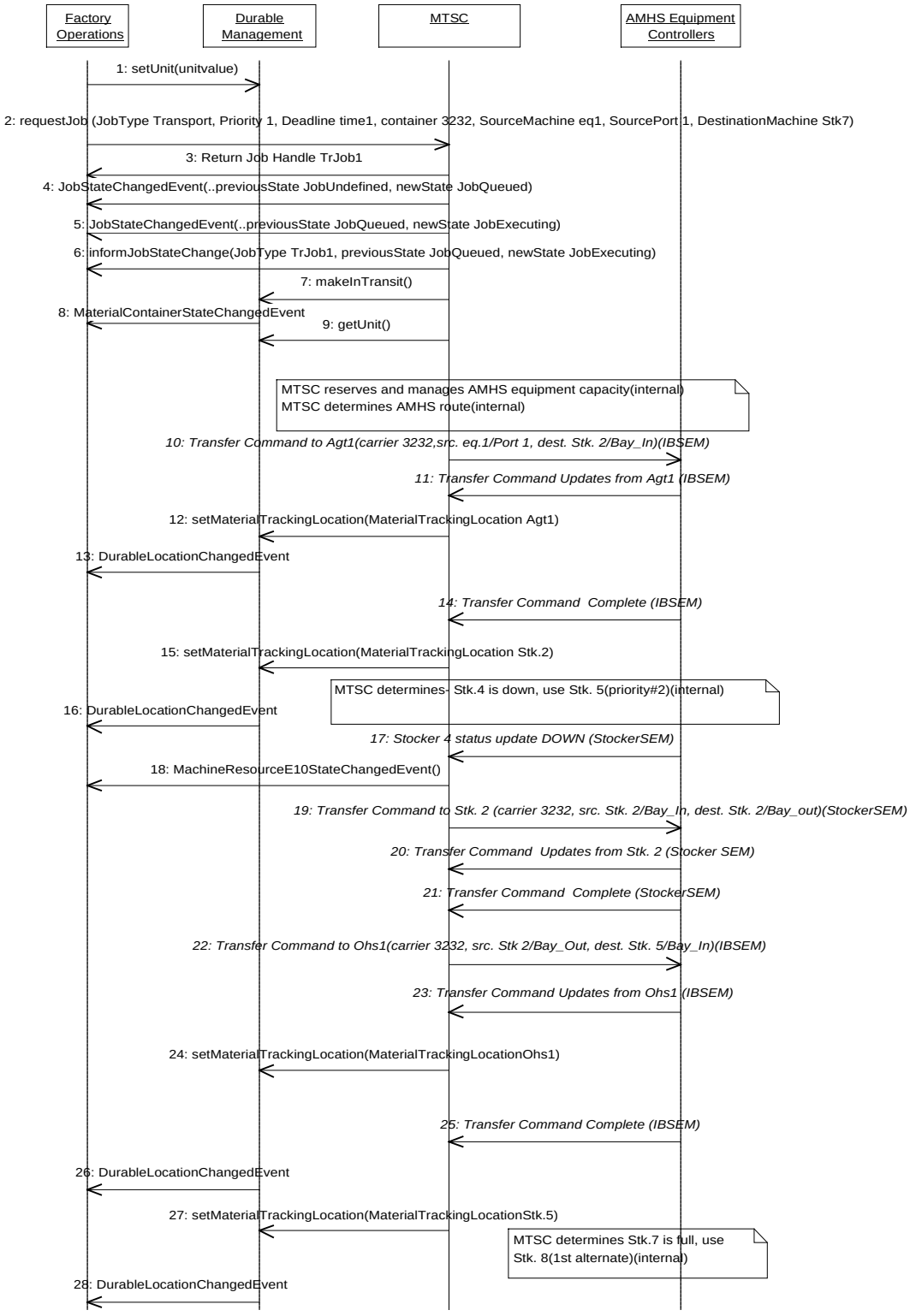

Figure R1-9
Scenario Case 4

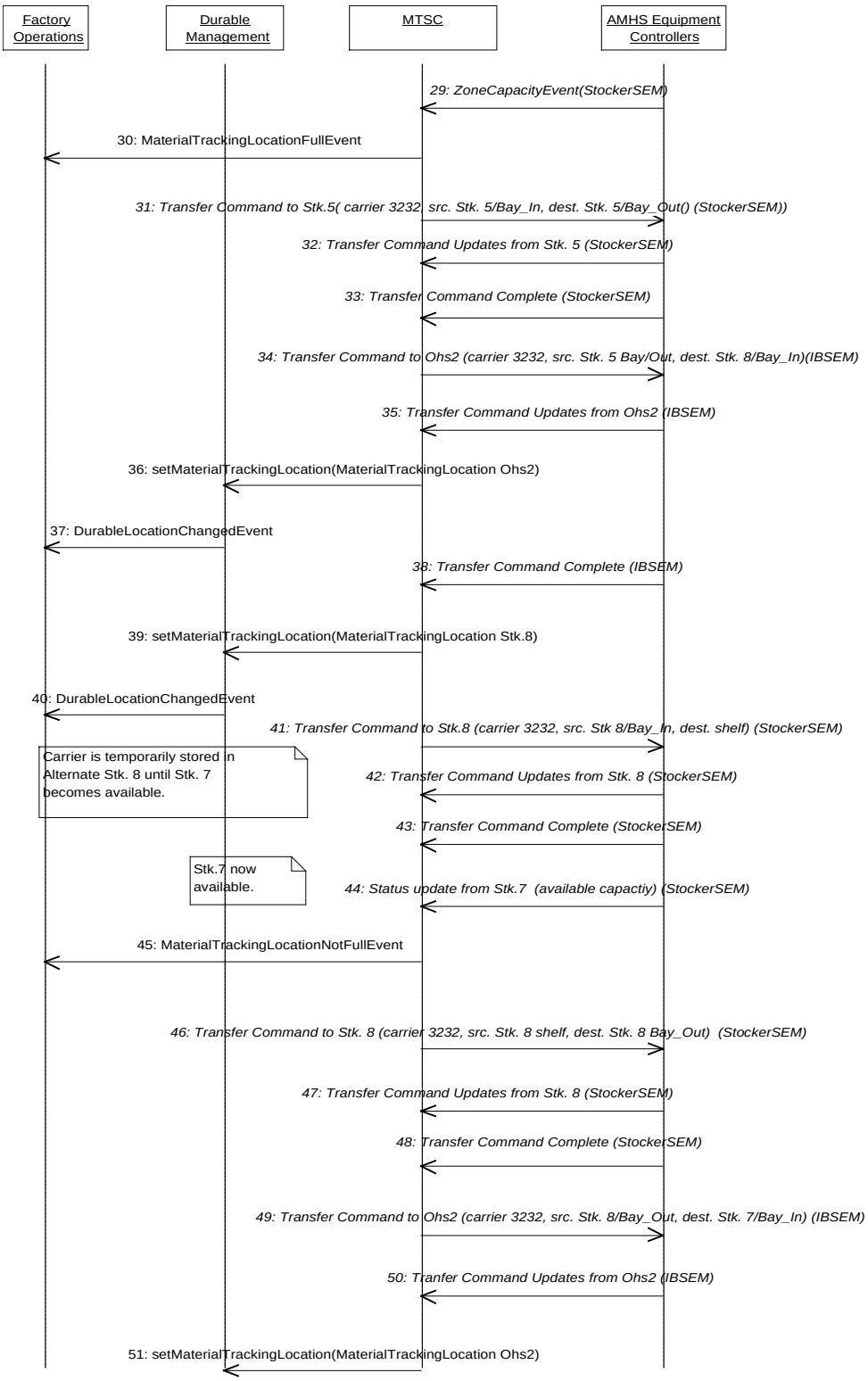

Figure R1-10
Scenario Case 4 (continued)

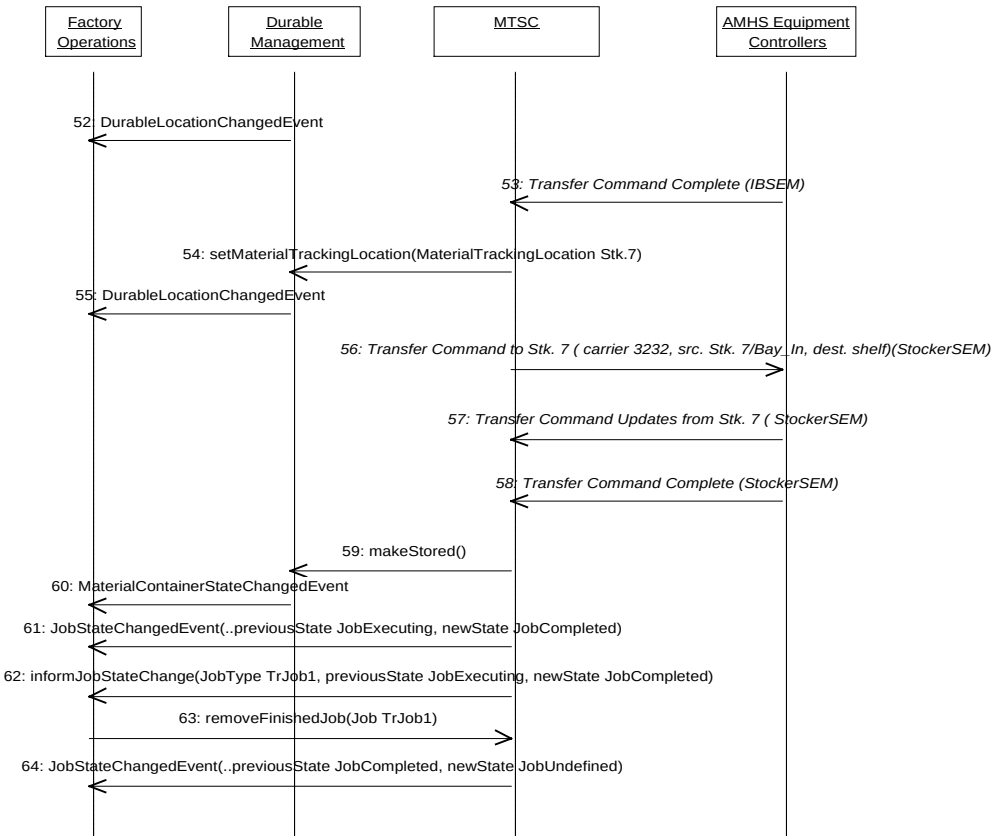

Figure R1-11
Scenario Case 4 (continued)

R1-2.5 *Scenario Five — Manual Carrier Input and Handling*

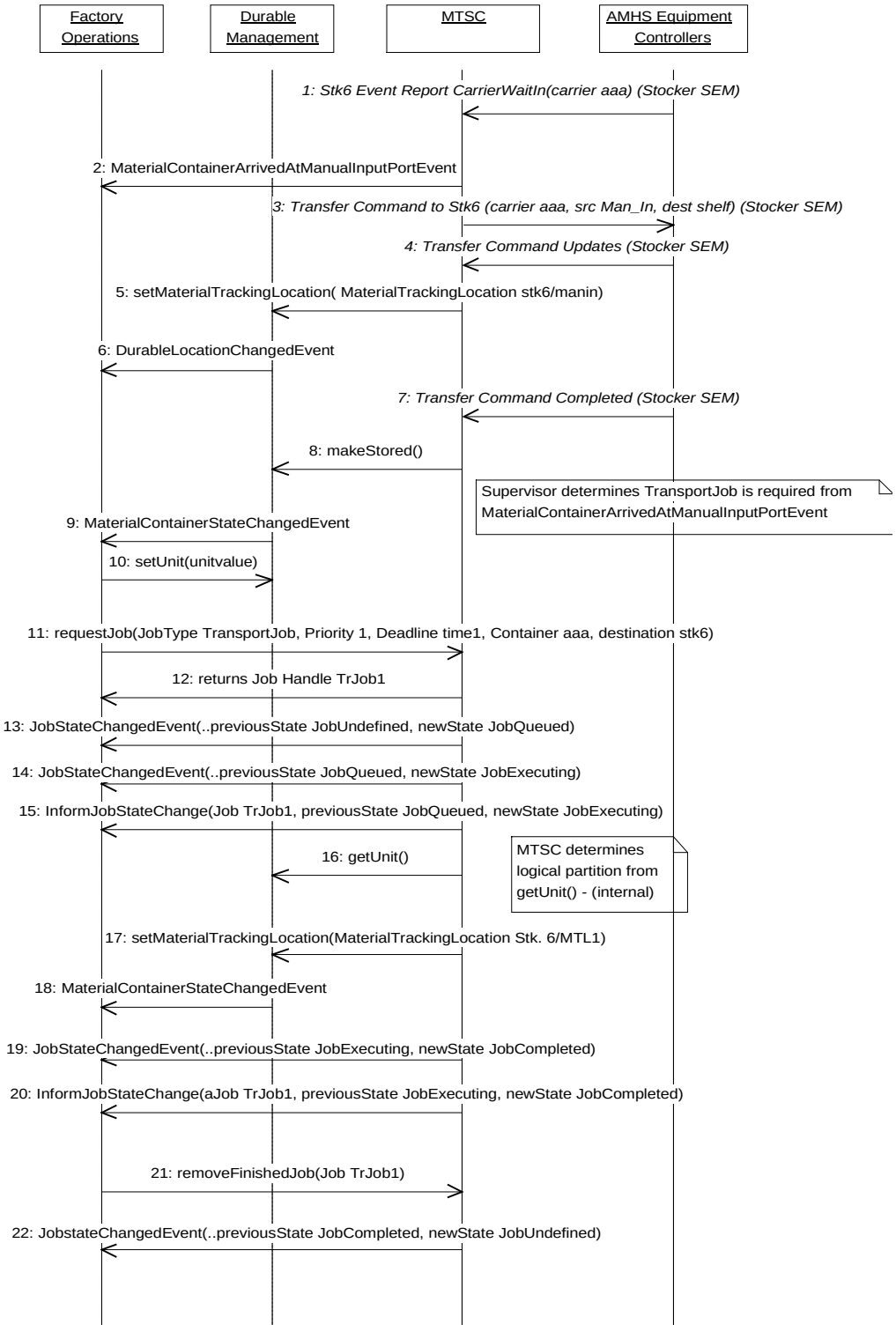

Figure R1-12
Scenario Case 5

R1-2.6 *Scenario Six — AMHS Equipment Monitoring*

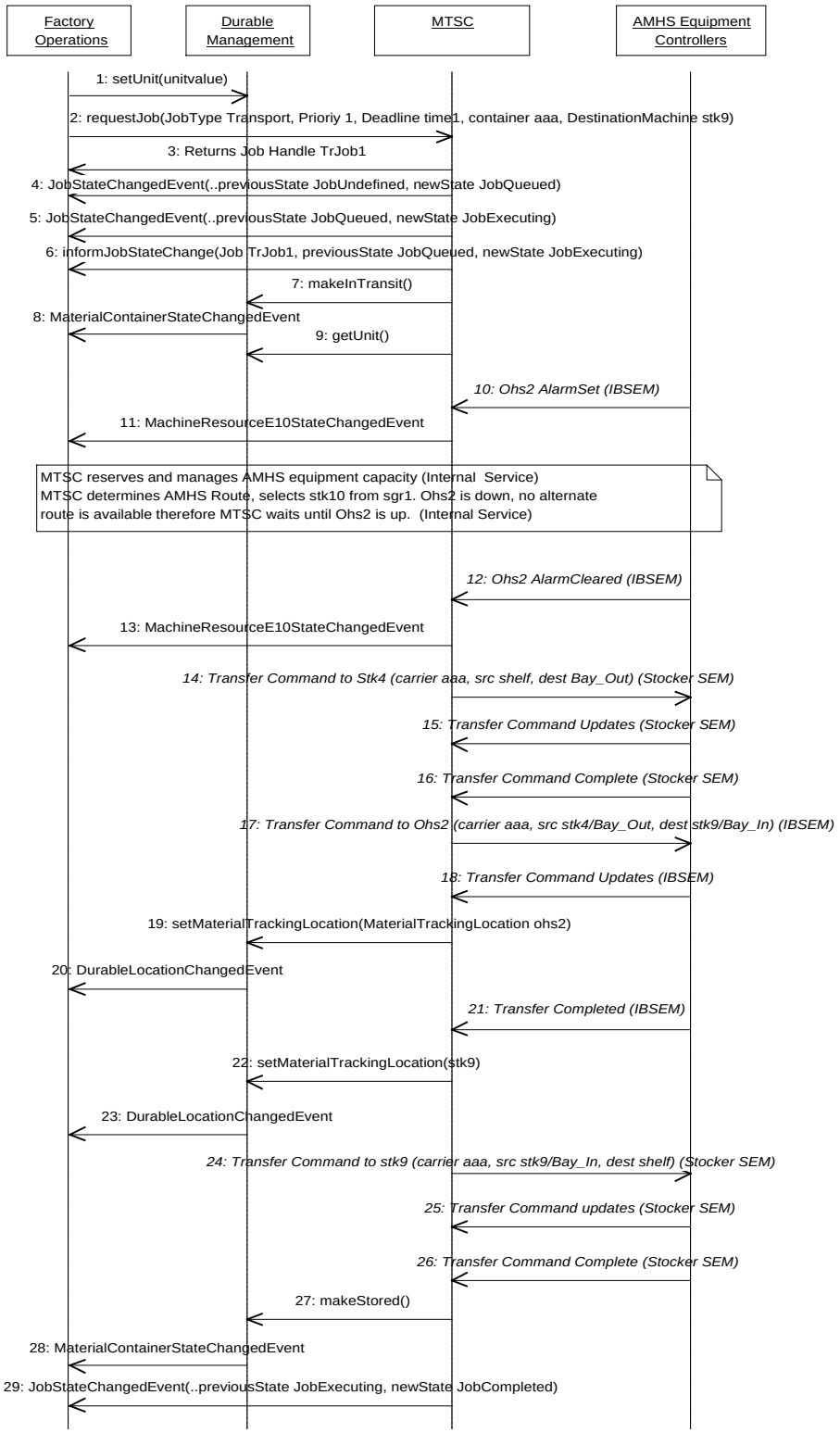

Figure R1-13
Scenario Case 6

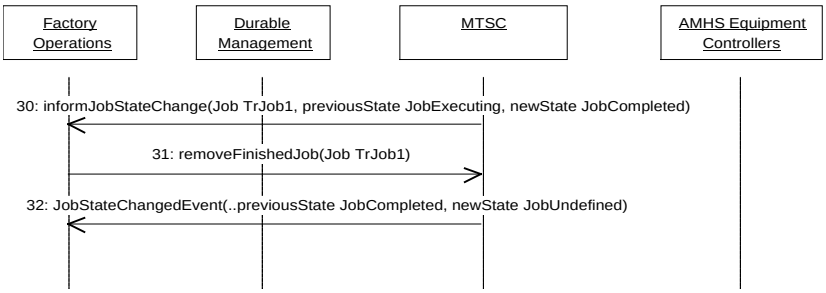

Figure R1-14
Scenario Case 6 (continued)

R1-2.7 *Scenario 7 — Batch Delivery to Internal Buffer Equipment*

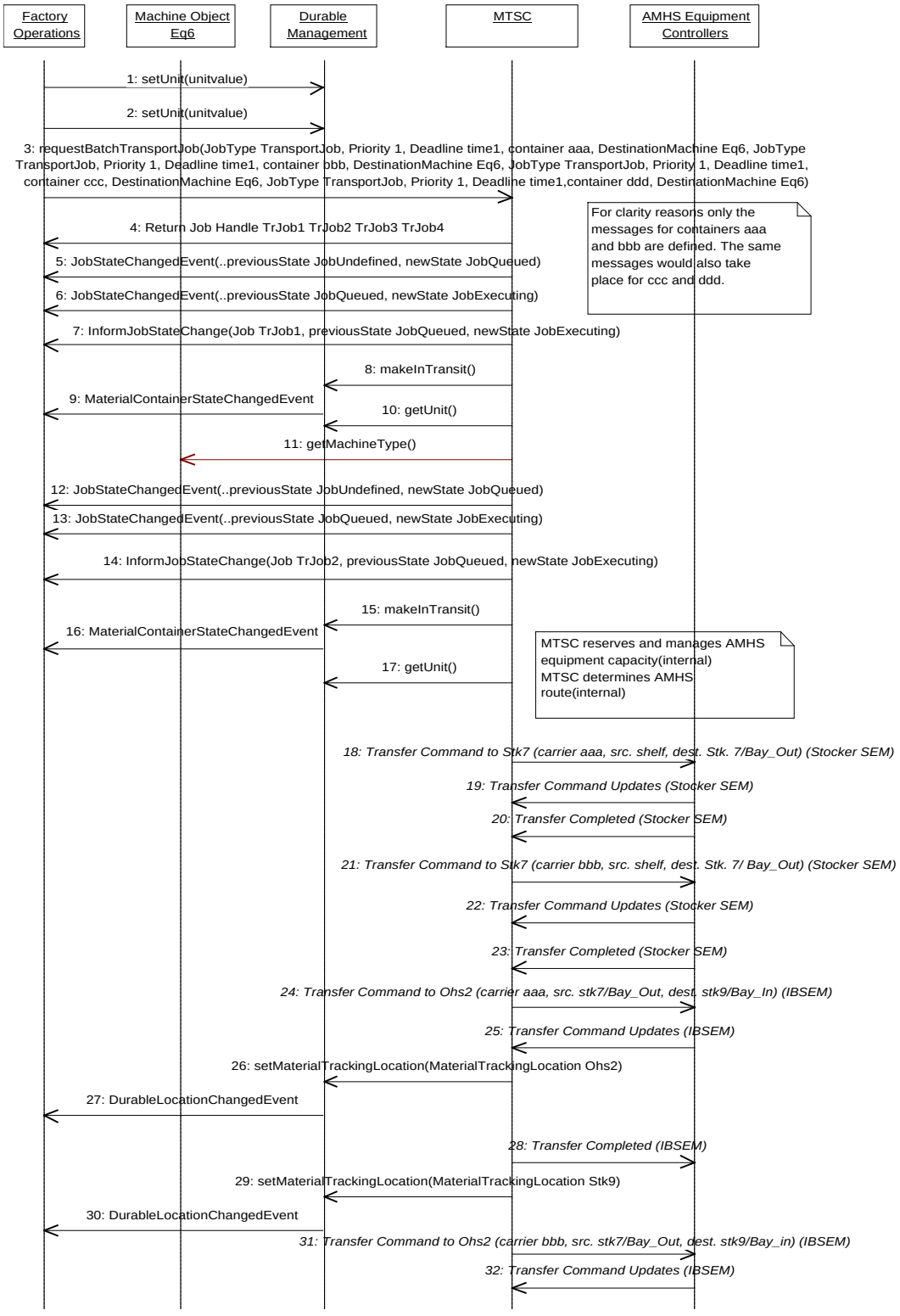

Figure R1-15
Scenario Case 7

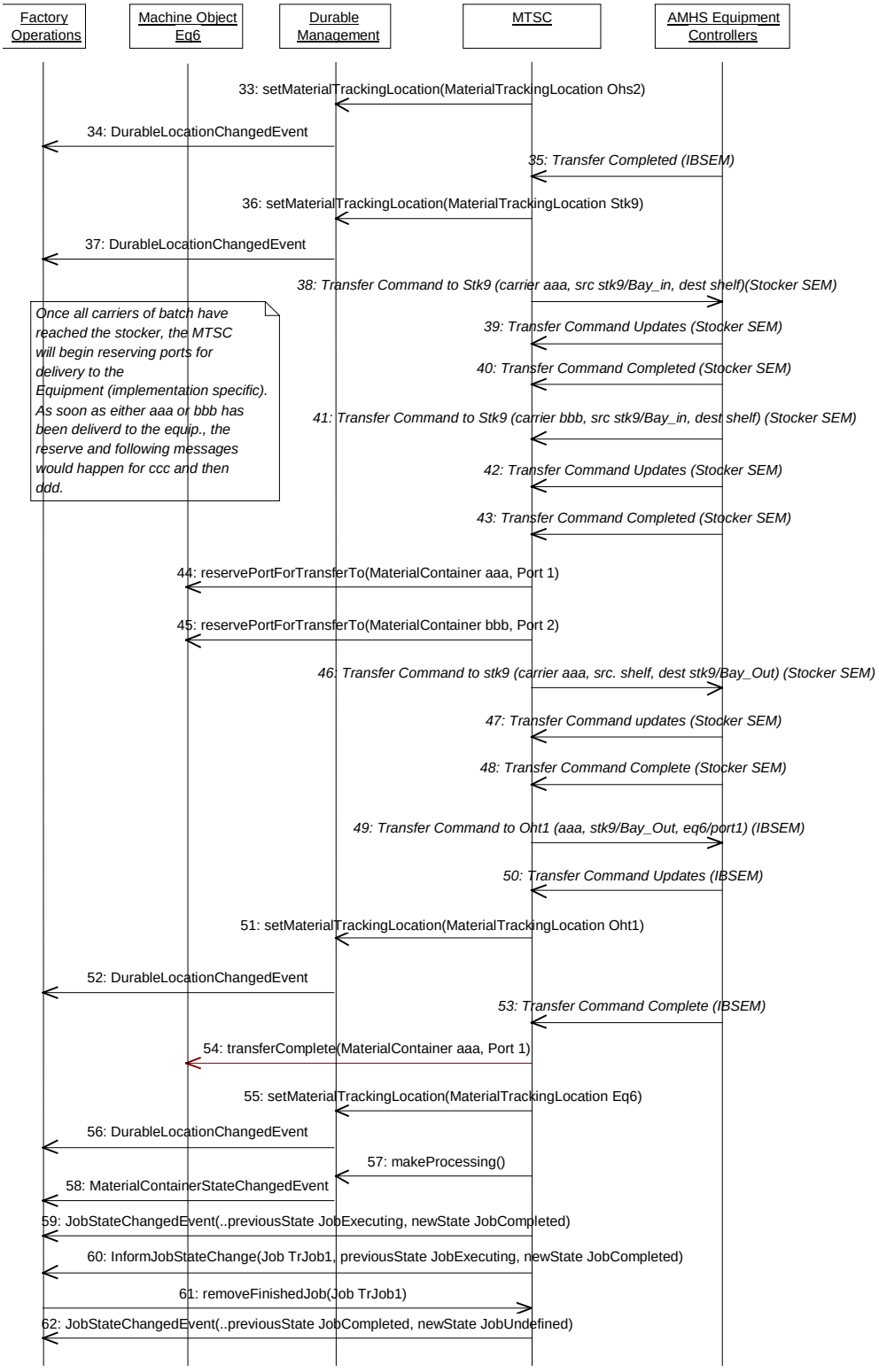

Figure R1-16
Scenario Case 7 (continued)

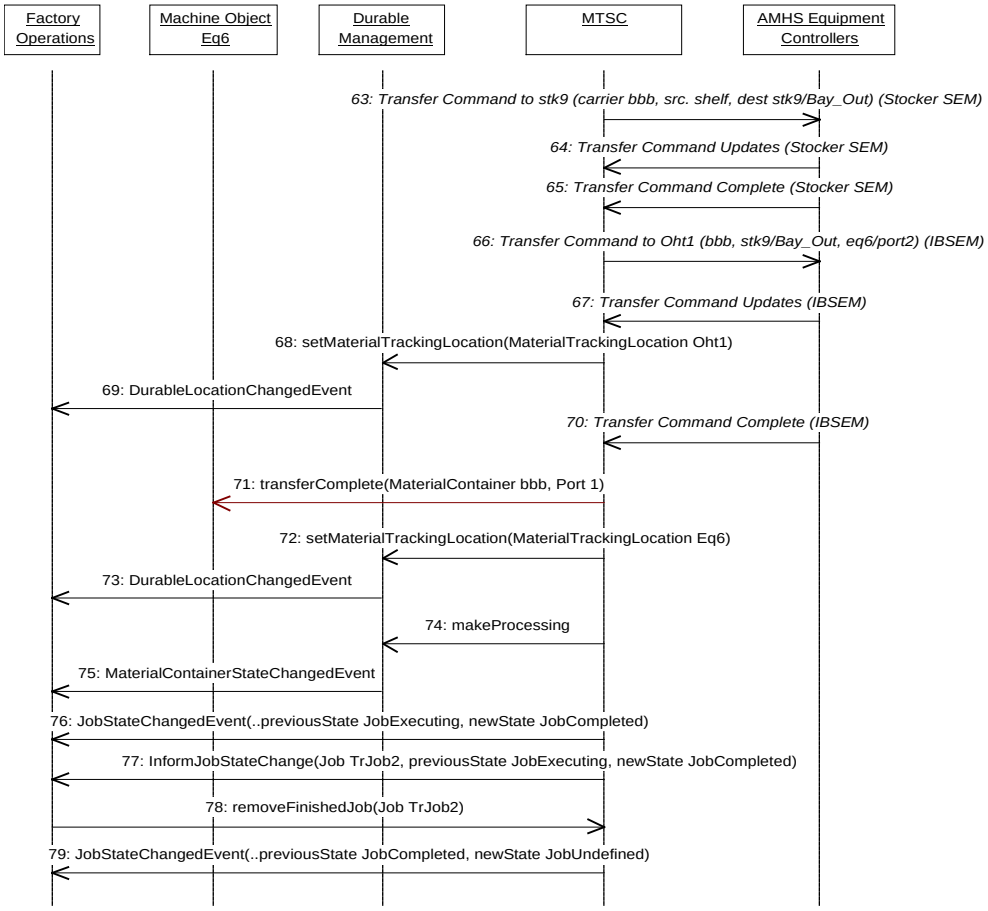

Figure R1-17
Scenario Case 7 (continued)

R1-2.8 *Scenario Eight — Abort Job Handling*

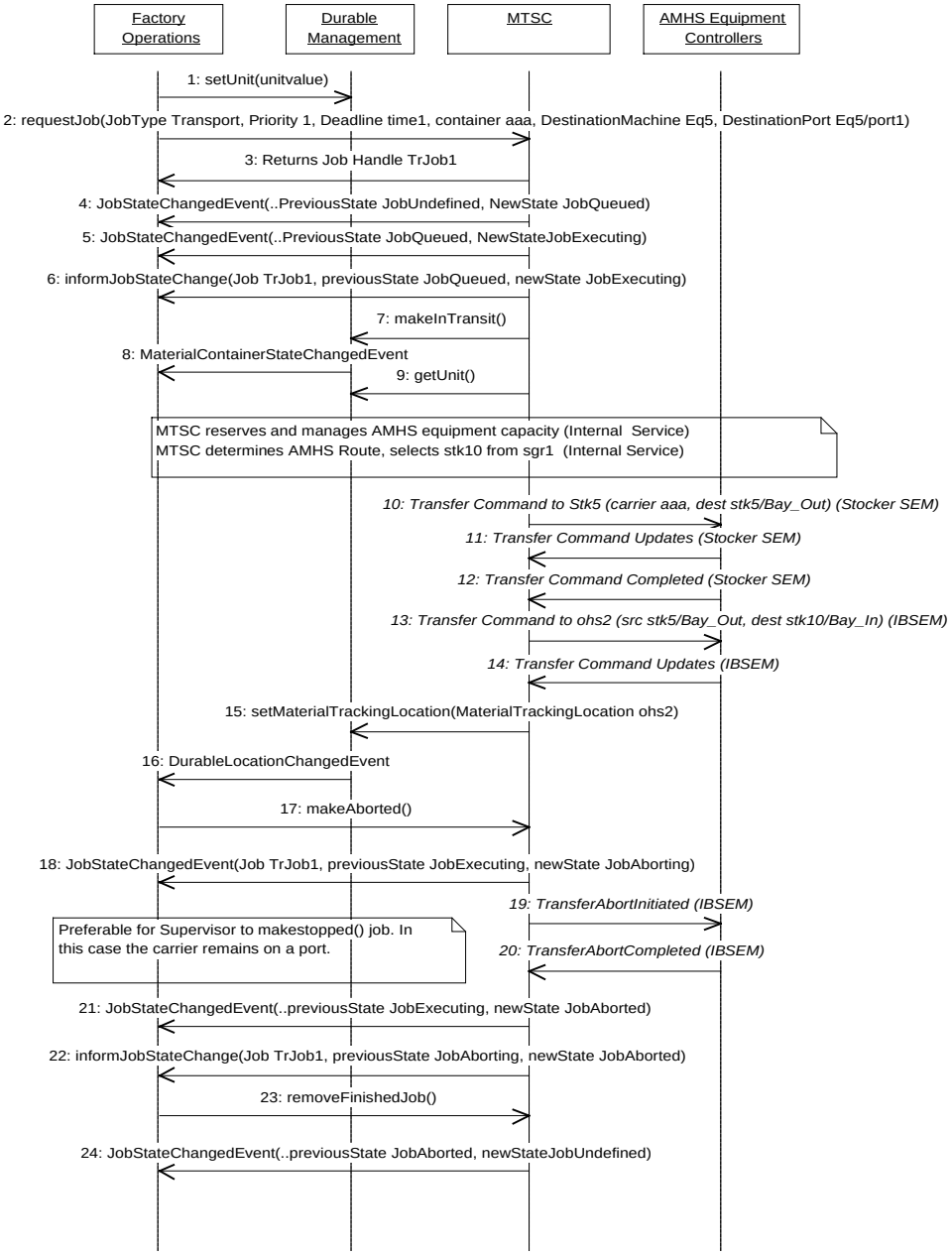

Figure R1-18
Scenario Case 8

R1-2.9 *Scenario Nine — Modify Job Handling*

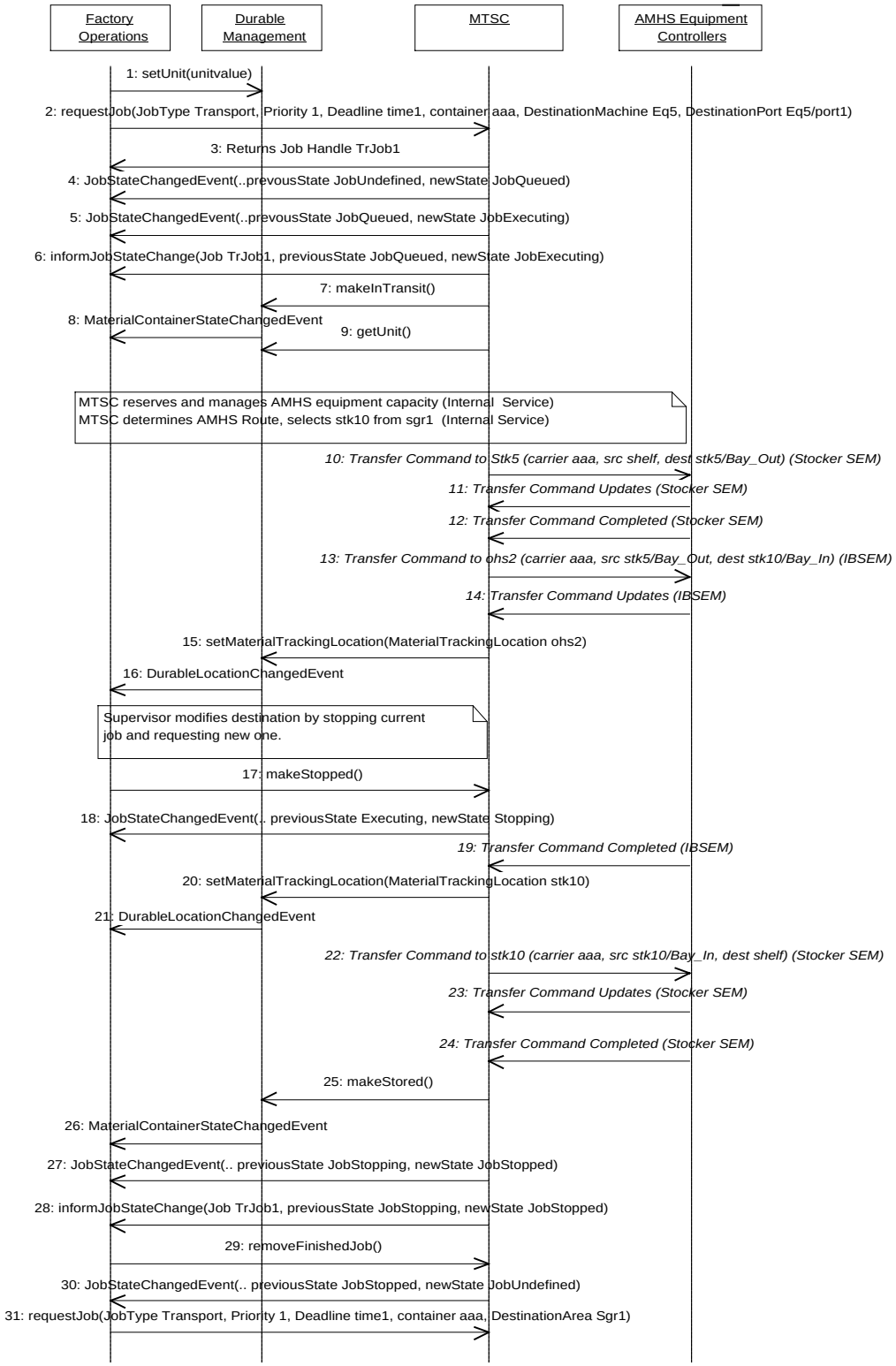

Figure R1-19
Scenario Case 9

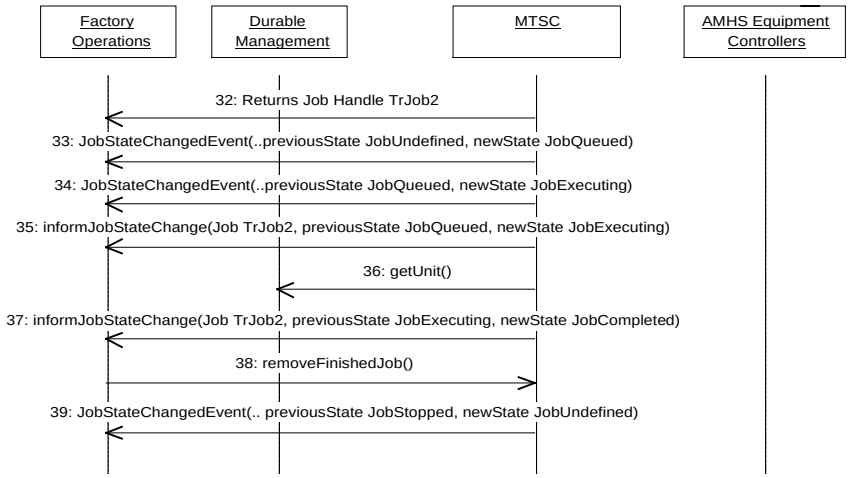

Figure R1-20
Scenario Case 9 (continued)

R1-2.10 *Scenario Ten — Material Transport and Storage Component Startup*

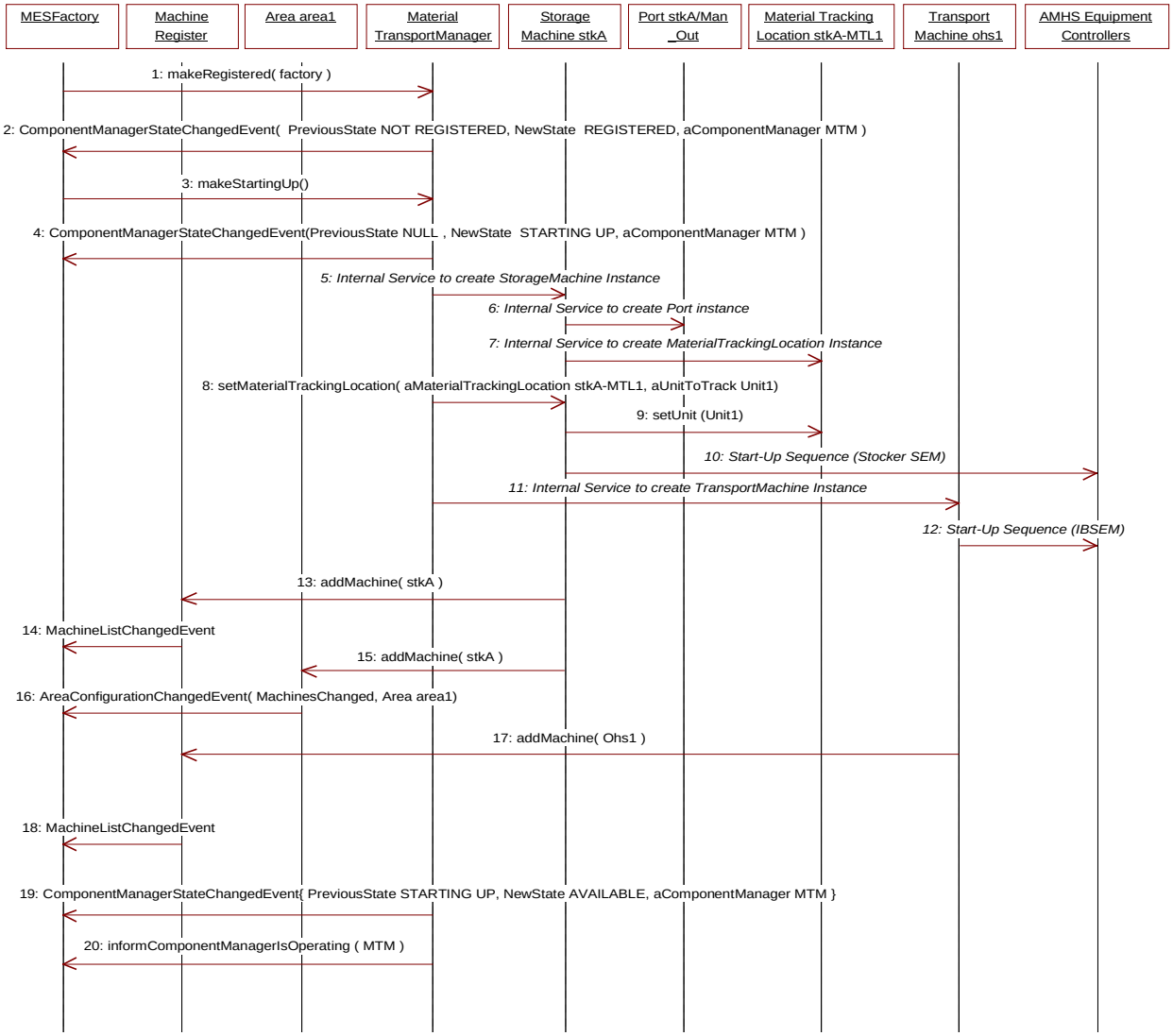

Figure R1-21
Scenario Case 10

R1-2.11 *Scenario Eleven — Manual Interruption of Delivery*

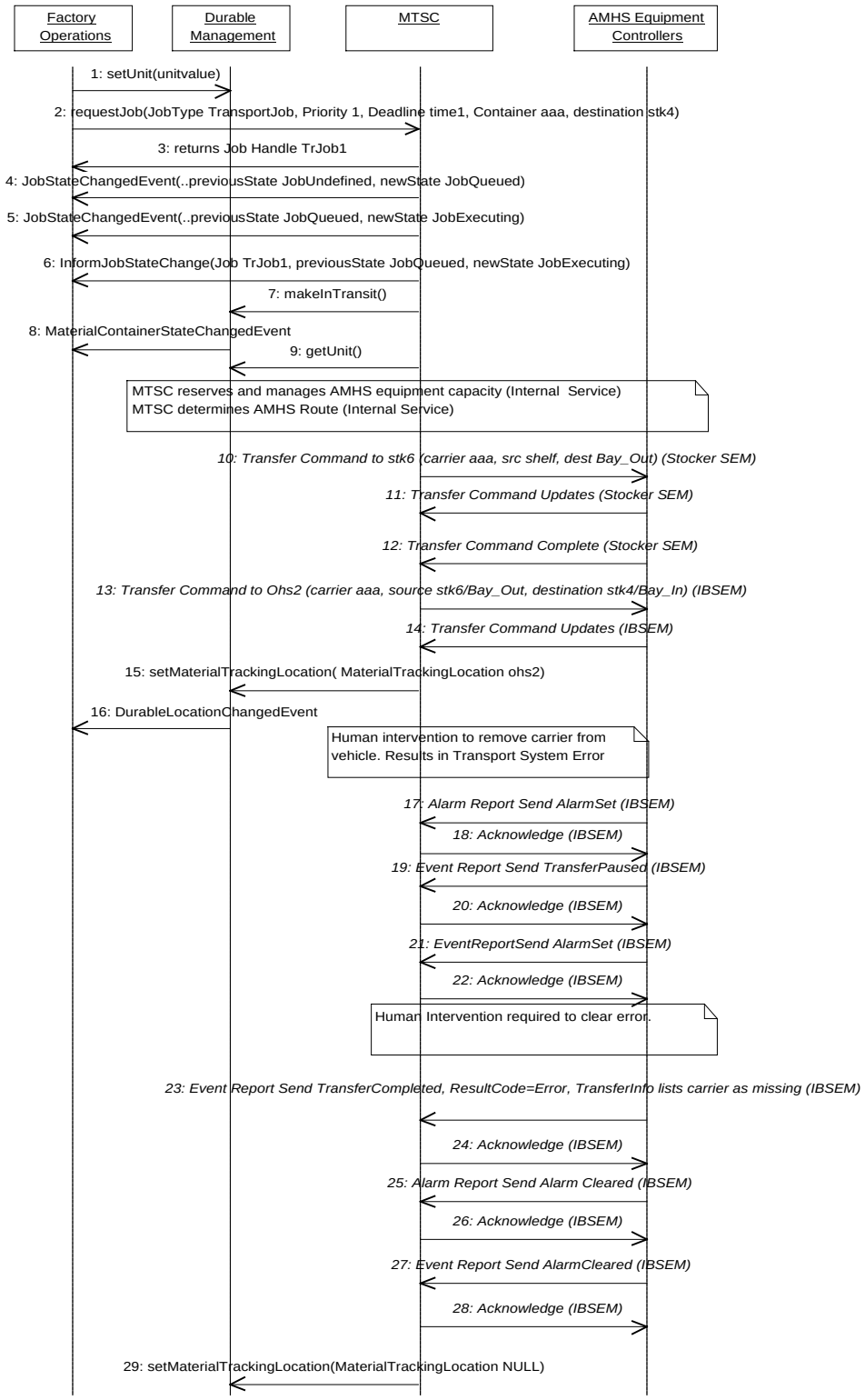

Figure R1-22
Scenario Case 11

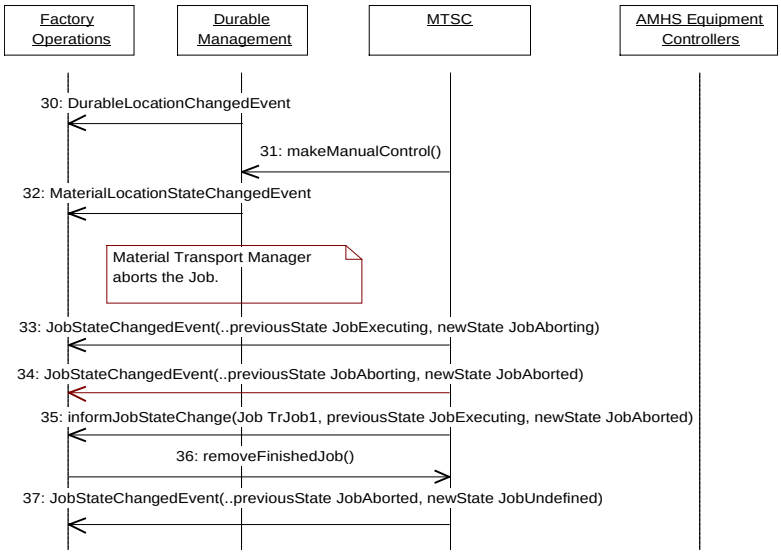

Figure R1-23
Scenario Case 11 (continued)

R1-2.12 *Scenario 12 — Tool to Tool Delivery*

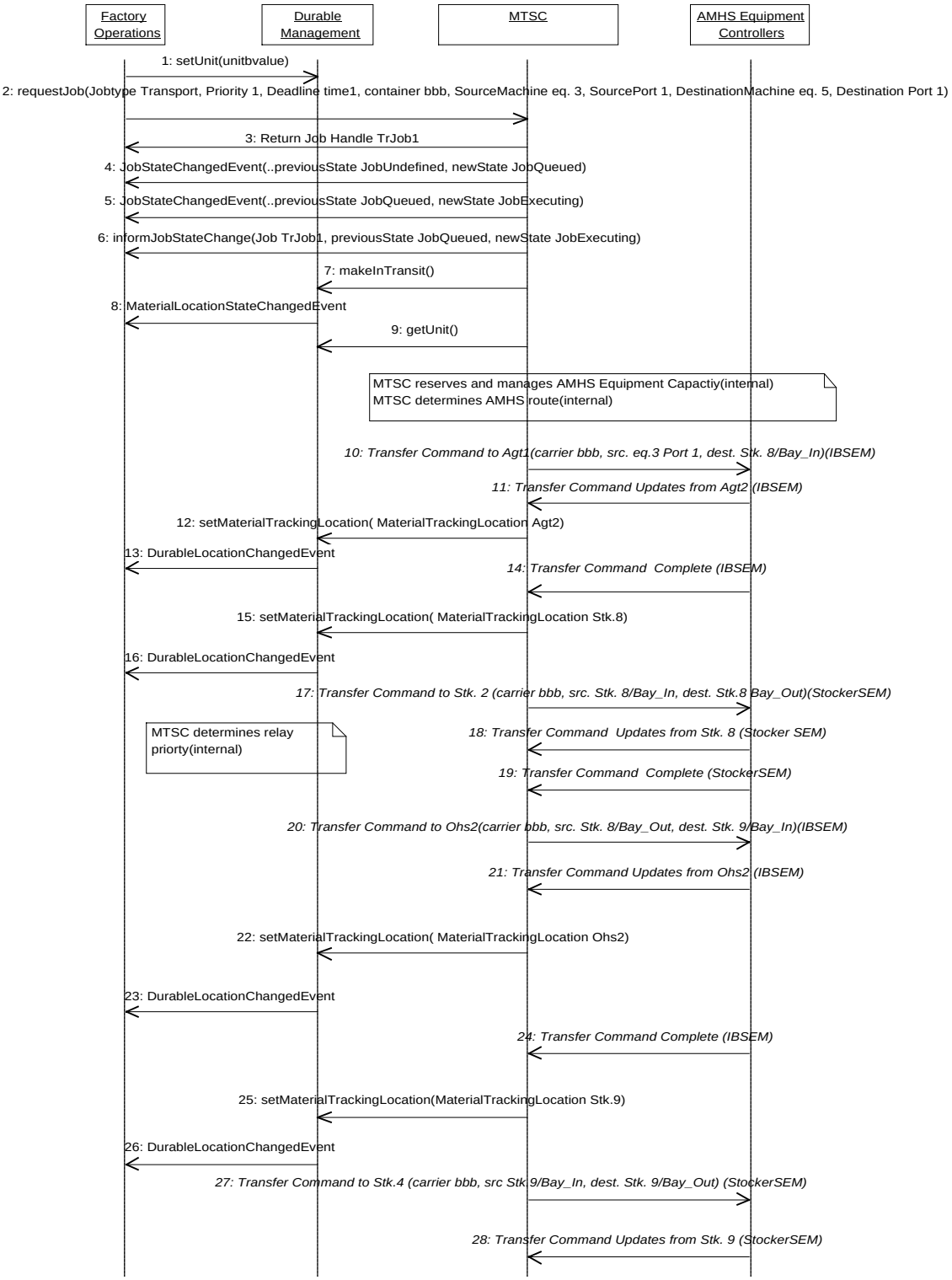

Figure R1-24
Scenario Case 12

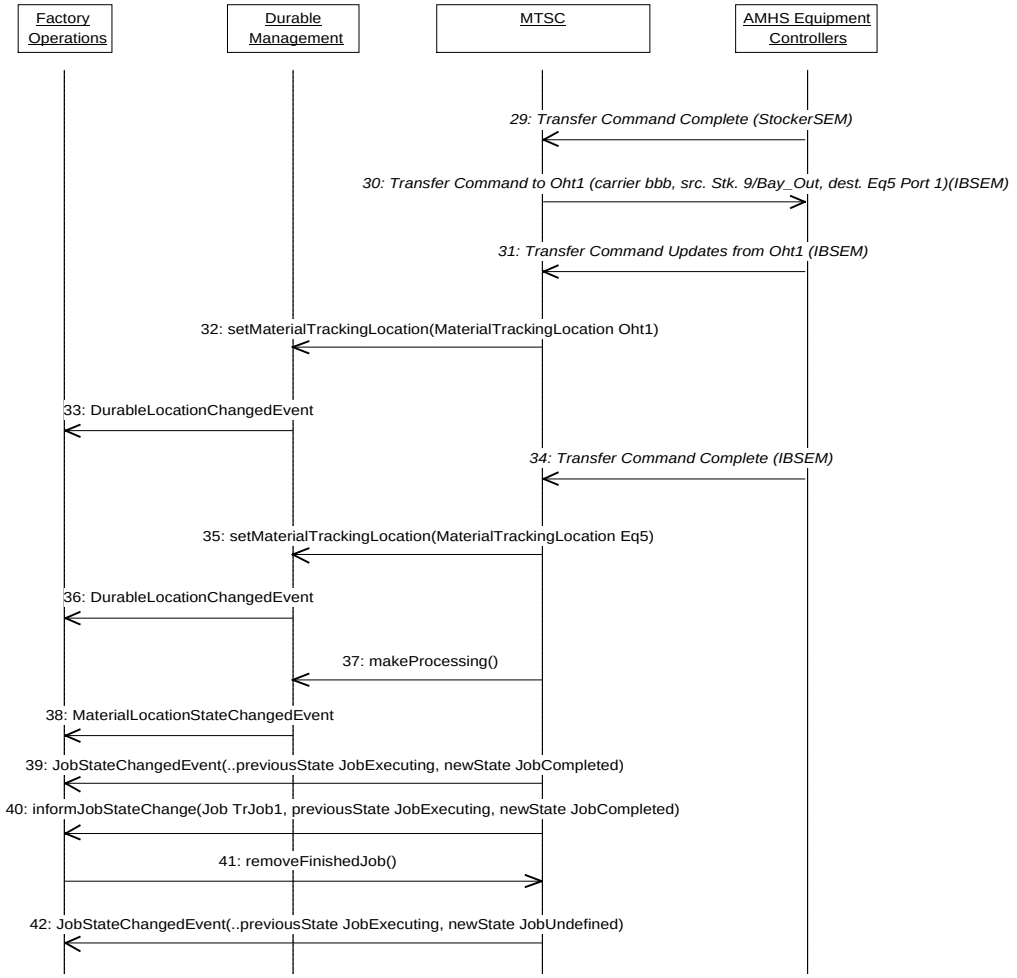

Figure R1-25
Scenario Case 12 (continued)

R1-2.13 *Scenario 13 — Interactions at an Individual Interface Level (including capacity reservation)*

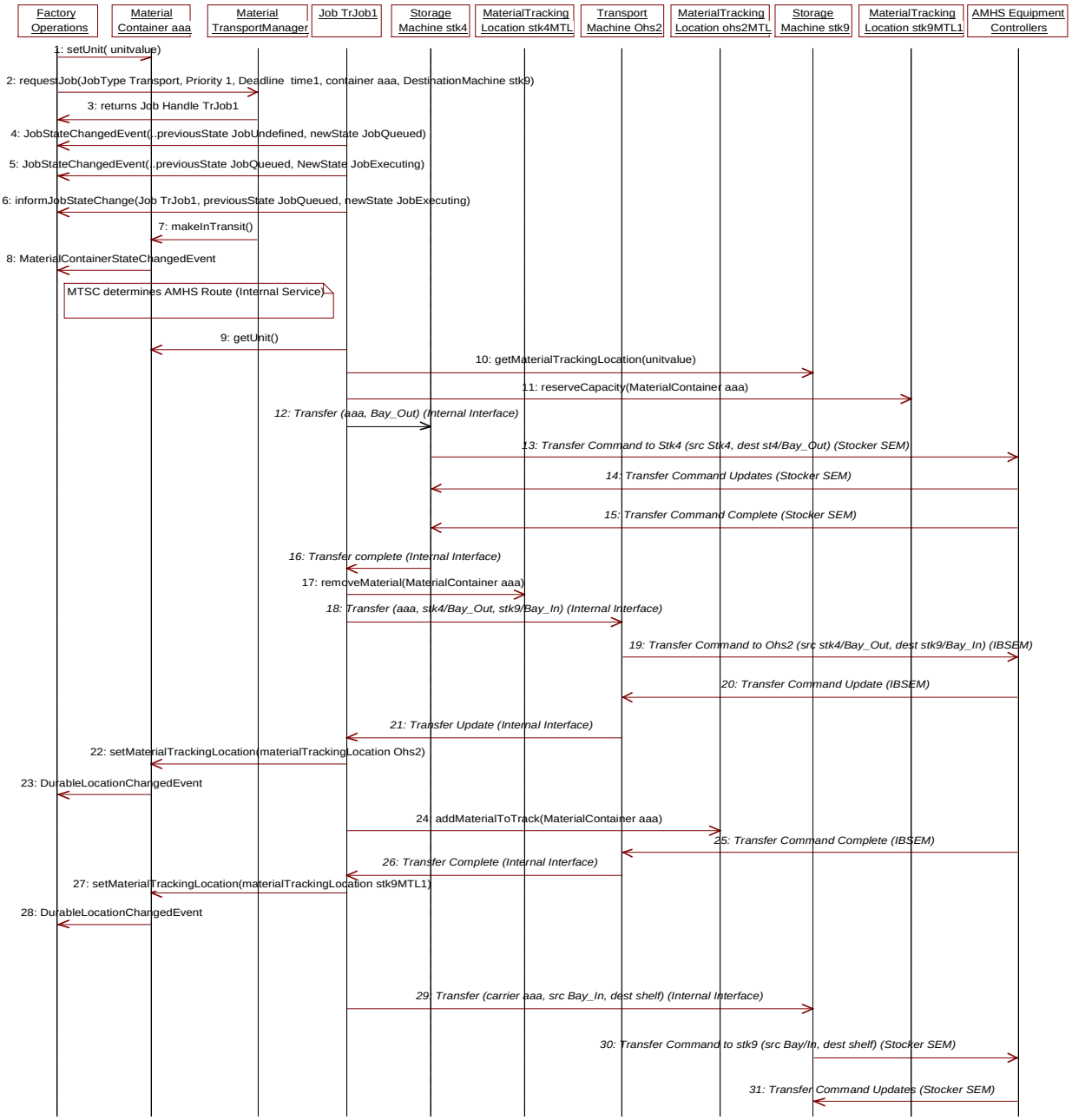

Figure R1-26
Scenario Case 13

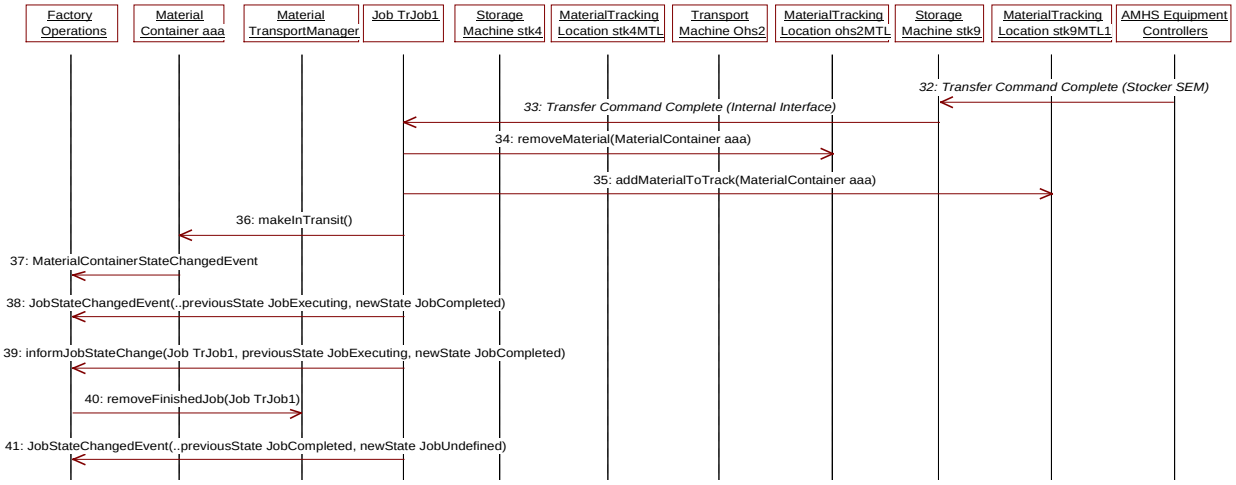

Figure R1-27
Scenario Case 13 (continued)

RELATED INFORMATION 2
LOGICAL PARTITION STORAGE

NOTE: This related information is not an official part of SEMI E10# and was derived from the work of the I300I/J300E AMHS workgroup accomplished during development of the proposed standard. This related information is included with the Material Transport and Storage Component specification to aid the readers in understanding the intent and use of the standard. This related information was approved for publication by full letter ballot procedures on January 14, 2000.

R2-1 The concept of Stocker storage via logical partitions has been implemented through the use of the MaterialTrackingLocation concept and interfaces. In the example below, the correlation between Durables and MaterialTrackingLocations are each object's unit attribute. First, the Factory Supervisor sets the Durable unit to a particular value based on user specifications. The MTSC then maps the Durable Unit to the appropriate MaterialTrackingLocation unit for the destination Storage Machine in order to determine the appropriate logical partition for that particular durable. It is not required that the Durable unit and MaterialTrackingLocation unit be identical in order for a mapping to occur, the MTSC may implement mapping logic in order to obtain various relationships between Durable units and MaterialTrackingLocation units.

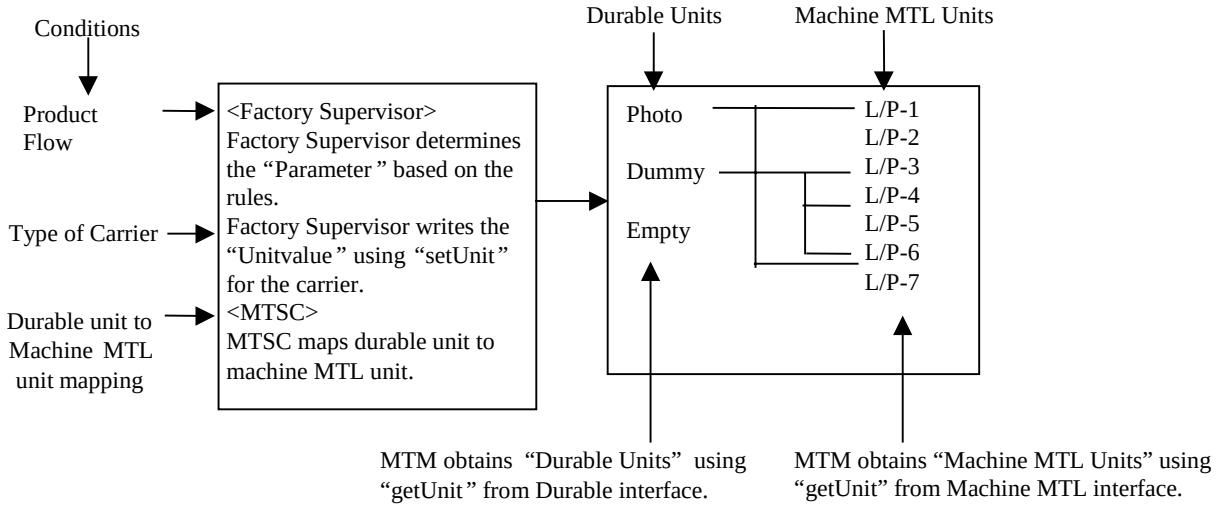

Figure R2-1
Storage in Logical Partitions

SEMI E105-0701
PROVISIONAL SPECIFICATION FOR CIM FRAMEWORK SCHEDULING COMPONENT

This specification was technically approved by the Global Information & Control Committee and is the direct responsibility of the North American Information & Control Committee. Current edition approved by the Japanese Regional Standards Committee on February 1, 2001. Initially available at www.semi.org April 2001; to be published July 2001. Originally published October 2000.

1 Purpose

1.1 The Scheduling Component supports Factory Operations, Material Transport and Storage, Production Machine, and Equipment Tracking and Maintenance components by ordering, in time, jobs that process material on equipment, move material, and maintain equipment. The scheduler uses knowledge of product demand, equipment and material state, process flows, throughput bottlenecks, operational policy and constraints, and other information to recommend jobs that maximize effective utilization of factory resources to satisfy product demand and planned objectives.

1.2 Increased control over operations requires an active Scheduling Component that can respond to factory events and changes in state and dynamically adjust the schedules for material processing, material transport and equipment maintenance. The Scheduling Component can react to inventory levels of material in the factory to adjust priorities to minimize queue sizes and ensure that use of bottleneck equipment is optimized to keep WIP inventory levels at desired levels.

1.3 The Scheduling Component can minimize turn around time (TAT) by coordinating material transport (for substrates and durables) with processing to reduce equipment idle time. The Scheduling Component can sequence activities to minimize setup time. It can also respond to scheduled and unscheduled equipment down-time to minimize impact on turn around time. In addition to minimizing overall TAT, it can react to the priorities for urgent lots to move them through the process flow in the minimum possible time while adjusting the schedules of lower-priority lots that are impacted.

2 Scope

2.1 The primary run-time responsibilities of the Scheduling Component are to monitor resource and material state and apply scheduling and dispatching decision mechanisms to identify the next activity (dispatching) or sequence of activities (scheduling) for factory resources. The Scheduling Component includes an interface that supports both scheduling and dispatching. Figure 1 illustrates the interactions between the Scheduling Component and other components of the CIM Framework. This illustration does not reflect all of the many inputs to the Scheduling Component that are required to provide it with the current status of the factory resources.

2.2 As described in this standard, the Scheduling Component produces activity option and activity forecast lists. These lists are produced by combining the factory model and status information from other components with scheduling policies in the Scheduling Component. For example, the Scheduling Component combines data from the Specification Component on how products are made with status data from the Equipment and Product Management Components to give activity options for a machine.

2.3 The Factory Operations Component uses the Scheduling Component services to orchestrate the management of machines and production of lots. For example, when a tool becomes available, Factory Operations uses the Activity Options list to select the next lot to process on the tool. Factory Operations then works with other components to execute the production job for the lot on the machine. Executing the production job changes the state of lot and tool in other components. These status changes are then used by the Scheduling Component when it produces new activity lists.

2.4 The Scheduling Manager supports Factory Operations by providing an answer to questions like, “What is next for this material or resource?” The answer may be based on evaluation of current or future constraints and objectives. Although the dispatcher's output takes the form of a decision for the next activity for the target resource, the interface may also support manual scenarios by providing a list of prioritized activities from which the decision is selected.

2.5 The Scheduling Manager interface also provides forecasts of future activities projected to occur after the next activity. By simulating anticipated future activity sequencing and execution timing the scheduler can generate forecasts that predict future responses from the dispatcher for subsequent requests. These forecasts are subject to change as factory conditions change, but they offer the best current projection of future activity decisions.