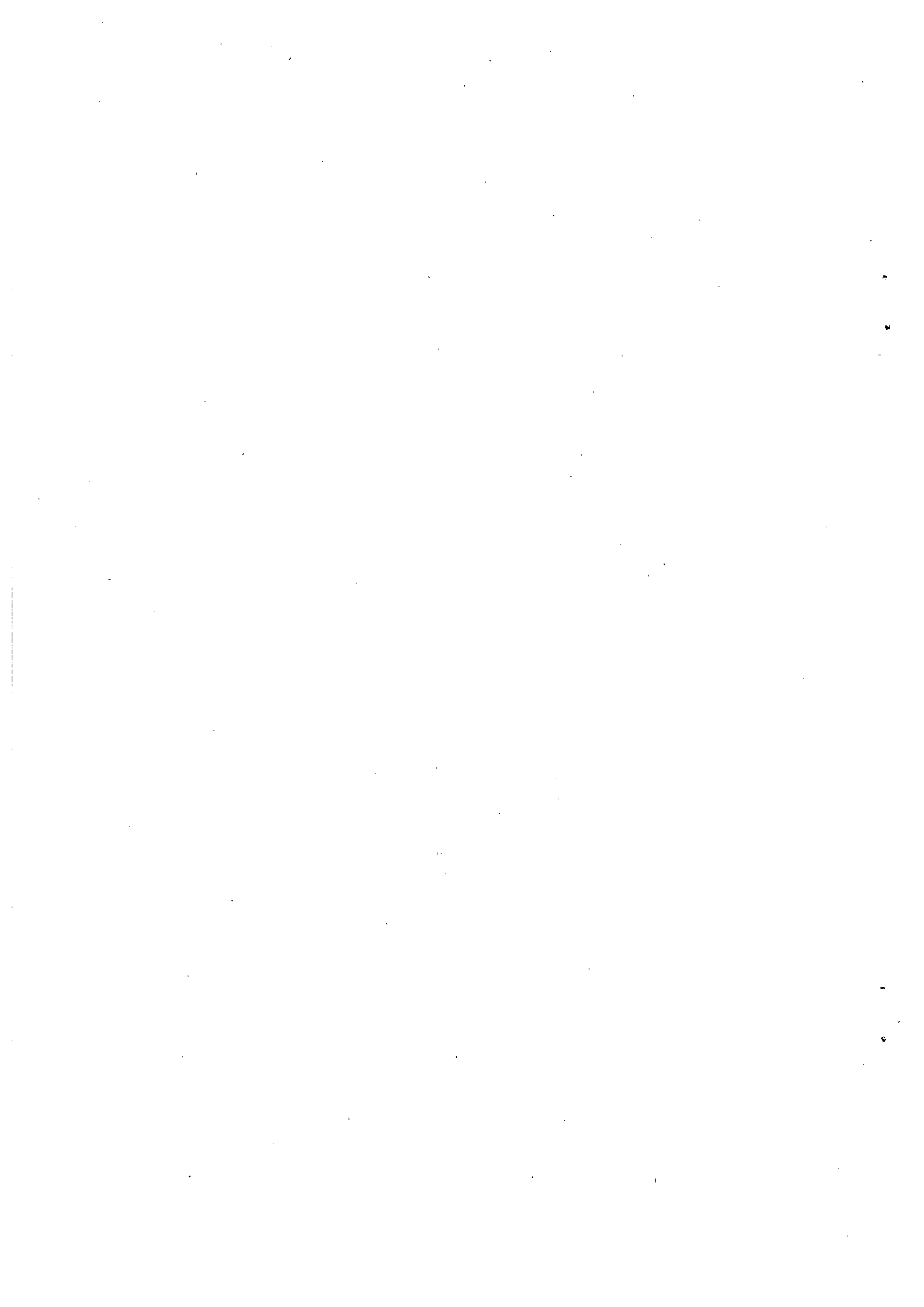
No. 999

# A UML Model of Agile Production Planning and Control System

by

T. L. Tsai and R. Sato

July 2002



# A UML model of agile production planning and control system

Tunglun Tsai*, Ryo Sato**

*Graduate School of Policy and Planning Sciences, University of Tsukuba

**Institute of Policy and Planning Sciences, University of Tsukuba

**Abstract**

A formulation of the agile production planning and control system (APPCS, for short) is given and its data model and methods are provided in this paper. The formulation consists of static model, planning and scheduling model, and control model. It is implemented as a simulator that can produce feasible production plan and schedule, and agilely response to uncertainties using safety lead time and safety stock for supplies.

**Keywords**

Agile production planning and control system; UML data model; Planning and scheduling; Uncertainty; Safety lead time

## 1. Introduction

Agile production planning and control system (APPCS) is a real time system that can respond to uncertainty immediately. It was proposed by Sato and Tsai (2002) as innovation to the traditional production planning, scheduling, and control system. It converts demands to a production and/or procurement requirement, generates schedules to realize the production requirement, and controls the processing of the schedules. This paper proposes a formulation of the APPCS that provides a structure and procedures for planning, scheduling, and event control according to the following system requirements.

1. Ability to make feasible production plan and schedule: A plan and schedule is a result of planning and scheduling process. A plan and schedule is feasible if it is realizable and executable in actual production. Sato and Tsai (2002) advocated that what is really problematic

** Institute of Policy and Planning Sciences, University of Tsukuba, Tennodai, 1-1-1, Tsukuba, Ibaraki, Japan.
** Tel.: +81-298-53-5543, fax: +81-298-55-3849
** rsato@shako.sk.tsukuba.ac.jp

for a plan is not the nervousness of a planning and control mechanism but the exact feasibility of the plan. A feasible plan and schedule for a demand makes the demand management be capable of promising an order-delivery-date precisely. One of the weaknesses of MRP pointed out by Silver et al. (1998) that it uses a lead time specified by a user for each part and since queues of the components of a part can be formed in front of work centers, actual lead time varies. A feasible production plan and schedule hence becomes the first prerequisite of APPCS.

2. Flexibility of planning and scheduling: The traditional planning and scheduling follows the natural flow of decisions in an organization (Shobrys and White, 2002). It first plans all the materials and then allocates capacity to those materials. However, Goldratt and Cox (1986) suggest that scheduling an important resource first can avoid the occurrence of bottlenecks and improve the throughput of production. Thus, APPCS must provide a flexible procedure, which is specified by parameters, to efficiently allocate materials and resources to a demand.
3. Agile response to uncertainties: Uncertainty becomes inevitable in modern times. It can be regarded as an important message for the production system to keep up with the changing market requirements. Sanchez and Nagi (2001) regards a planning, scheduling, and control system that is able to re-schedule or recover from many uncertainties of the market as a further research of agile manufacturing system. However, Silver et al. (1998) indicated that the frequent updates from the planning process leads to a poor communication between the planning department and the manufacturing department. Hence, APPCS is requested to provide a structure to immediately response to uncertainty and cope with the uncertainty with less conflict.

A similar structure was proposed by Hegedous and Hopp (2001) in calculating optimized safety lead time for a purchase part. They modeled the whole production process as a single machine. Since different production processes are used for different products, it is plausible that the effect of buffers on a whole production process varies among the structures of the process. Thus, it is advantageous for setting of buffers to consider with BOM, routings and resources.

Tu (1997) gives a problem domain to production planning and control in a virtual one-of-a-kind production (OKP) company under uncertainty. The OKP offers 'one' product that belongs to 'a-kind' to an individual customer according to specific requirements. The problem domain shows (1) real-time monitoring and control of production progress, (2) a control structure that can be flexibly extended to cope with the uncertainties, and (3) an adaptive production scheduling structure and the algorithms are needed, etc. The APPCS acts as a model of the physical process for the problem domain.

To satisfy the above requirements, APPCS uses control structure of network, which is composed of components and links. A component plays a role of a provider, a requester, or both. A link connecting a provider with a requester means that the provider offers components to the requester for production. A demand is converted to the network structure by executing a set of planning processes and scheduling processes. Each planning or scheduling process is processed for a component. The planning process is to link a set of provider components with a requester component. The scheduling process is to generate schedules by allocating finite capacity of specified resource to the operations of the component.

A unique sequence of scheduling processes and planning process for a demand is determined by specifying some priority rules and constraints. Choosing next component for planning or scheduling, and choosing an apt resource among candidates are such rules. The priority rules and constrains are parameters to determine a flexible procedure of planning and scheduling. A demand is backwardly scheduled from its promised delivery epoch. If the result shows the demand must start from the past, then a forward scheduling approach is substituted to plan and schedule the demand from the present time. A feasible production plan and schedule of the demand is thus generated.

The components in the planned network are executed as it was planned. The production control of APPCS is to manage and monitor the execution of the on-processing components under

uncertainties. An event will cause components of the network unattainable. Except for the on-processing components, the planned components that link to the components directly or indirectly are cancelled and rescheduled to cope with the uncertainty. The network structure is designed to be able to do planning and scheduling over and over as any event is triggered. Thus, APPCS is capable of handling uncertainties.

A data model for product data management and planning is available in Scheer (1994). This paper showed a basic augmentation so that APPCS is possible. The formulation of APPCS, called the APPCS model, is described by using the universal modeling language (UML) and illustrated with class diagrams. Section 2 gives an introduction to the UML and provides a sample class diagram. In section 3, a formulation of APPCS is presented as 3 sub-systems. They are static model, planning and scheduling model, and control model. The static model is to describe parts, BOM, operations, work centers, and resources. The planning and scheduling model serves as a control structure for planning and scheduling, and also production control. The control model defines uncertainties and their impacts on the other objects. In section 4, we implement a simulator for APPCS that uses safety lead time and safety stock under demand and supply uncertainties. Finally, a conclusion is provided in section 5.

## 2. Modeling methodology

The UML is a language for specifying, visualizing, constructing, and documenting the artifacts of software system, as well as for business modeling and other non-software systems (OMG 2000). The language is composed by Foundation, Behavioral elements, and Model Management packages. The package here is a grouping of model elements. The UML Foundation Package defines the constructs to abstract a static model. We follow the UML Foundation Package to model APPCS.

The UML Foundation Package is made up of three subpackages: the Core, the Extension Mechanisms, and the Data Types. The Core package defines the basic abstract and concrete constructs needed for the development of object models. The following constructs, which are part of

the UML Core Package, are used in modeling APPCS. Figure 1 demonstrates a class diagram of an example. The class diagram is a graph of model elements described as follows.

1. Class: A class is the descriptor for a set of objects with similar structure, behavior, and relationships (OMG 2000). An object instantiates operations from a class and assigns values to attributes of the class. In figure 1, *Human*, *Male*, *Female*, and *MarriageRegistry* are classes. *Mary* and *Smith* are objects instantiated from the *Human* class.
2. Attribute: An attribute is a named slot within a class that describes a range of values that instances of the class may hold (OMG 2000). In the diagram, *name*, *weight*, *age*, and *marry* are attributes of the *Human* class. The object of the *Human* class has values of its attributes. For example, the fact that the weight of *Mary* object equals 60 is written as *Mary.weight=60.*
3. Method: A method is the implementation of an operation. It specifies the algorithm or procedure that affects the results of an operation (OMG 2000). In the sample, a method, *eat*(), is defined in the *Human* class. The method is instantiated as an operation of *Mary* object such that *Mary.eat*() could increases *Mary.weight.*
4. Association: An association declares a connection between instances of the associated classes (OMG 2000). The instance of an association is a link, which serves as a reference to the objects it connects. In figure 1, *MarriageRegistry* and *Birth* are associations. The *MarriageRegistry* regulates how object of the *Male* class is related to object of the *Female* class. A multiplicity rule limits the number of target objects a source object can connect. For example, a male can marry 0 or many (0..*) females and vice versa in one's life. The *Birth* association reveals that a human object must connect to at least a marriage registry.
5. AssociationClass: An association class is an association that is also a class. It not only connects a set of classes but also defines a set of attributes that belong to the relationship itself and not any of the classes (OMG 2000). The *MarriageRegistry* is an association class as shown in figure 1.
6. Generalization: A generalization is a taxonomic relationship between a more general element

and a more specific element. The more specific element is fully consistent with the more general element and may contain additional information (OMG 2000). Figure 1 shows that *Male* class and *Female* class are two generations of the *Human* class.

[Insert figure 1 about here]

The Extension Mechanisms package specifies how specific UML model elements are customized and extended with new semantics (OMG 2000). A constraint is one of the extension mechanisms. It specifies a semantic condition among model elements. A system is valid only when all of the constrains attached to it are proved to be true. In figure 1, "any male can only marry to a female who is single, and vice versa" is defined for the *MarriageRegistry* association. The Data Type package specifies data types that are used to define UML itself.

## 3. A formulation of APPCS by UML

### 3.1. Static model

A part is a finished product, an assembly, or a raw material. A part is served as an input to or an output from a manufacturing process. A manufacturing process is a process of sales, production, or procurement. A raw material is procured by releasing purchase orders to venders. A finished product or an assembly is produced by its components, which could be other assemblies or raw materials. A finished product is sold to the customers. Figure 2, provided by Vollman et al. (1997), shows a product structure diagram of a snow shovel. In the figure, a rectangle represents a part, a solid line with a number denotes a usage of production, a dot line expresses the procurement process, and an arrow line indicates sales process.

[Insert figure 2 about here]

The production of a part is through a routing of operations that use components of the part. For example, the snow shovel is produced by (10) combining one top handle assembly with one shaft by 4 nail-62s to make one work-in-process (WIP) A, further by (20) fixing one scoop-shaft connector with one scoop assembly by 4 rivets to make one WIP B, and finally by (30) assembling

A into B. Each operation is assigned to a work center for processing. The total processing time is estimated by a summation of setup time, and processing time. The processing time is calculated by multiplying unit process time with processing quantities.

A resource with a qualified skill is enrolled as a member of a work center that needs the skill. A work center can enroll numbers of resources depending on the total capacity requirement. One of the resources enrolled in a work center is dynamically selected by an operation assigned to the work center when runs a planning and scheduling. The planning and scheduling process will be explained in section 3.3 in further detail.

The capacity of a resource is finite and managed by shifts. A shift, represents an interval of working hours, is specified for a resource on the planning horizon. The planning horizon indicates a time toward the future for which production planning is made. By assuming finite loading, the planning and scheduling uses the capacity of a resource only inside its shifts. Figure 3 demonstrates instances of operation, work center, resource, and shift, and mutual links among them. For example, an operation (Combining) is processed at the work center (Drive), the work center enrolled two resources (A, B, and C) with their respective shift.

[Insert figure 3 about here]

We compile and abstract the above descriptions as a static model of APPCS as follows. A class diagram of the static model is shown in figure 4.

[Insert figure 4 about here]

1. The part objects are abstracted as a *Part* class. The attributes of the *Part* class include *leadTime* and *inventory*. The *leadTime* of a part preserves the estimated time for producing or procuring a part regardless of its request quantity. The *inventory* is an initial stock that is possessed by the production system from the beginning.

2. The hierarchical relationships among parts can be abstracted as a *Bom* association with its both ends connecting with the *Part* class. A link of the *Bom* class in figure 2 is a solid line represents a usage from a part to one of its components. Hence the *Bom* association is also a class with attributes of *part*, *childPart*, and *usage*. A bill of materials (BOM) of a part is a set of the links whose 'part' end connects to the part. The product structure diagram, as shown in figure 2, of a part can be attained by spreading out its BOM downwardly and repeatedly until all raw materials of the part are reached. A constraint is necessary to prevent the product structure from an endless loop of BOM explosion.
3. A routing of operations of a part is abstracted as an *Operation* class and the *Part* class is a composite of it. The attributes of the *Operation* class are *oprId*, *setupTime*, *processTime*, and *workCenter*. The *oprId* is for identifying an order of operations. The total processing time processed at the work center *workCenter* for making $q$ pieces of a part equals to $setupTime + processTime \times q$.
4. The work centers and resources are modeled as a *WorkCenter* class and a *Resource* class with an *Enroll* association between them. The *Enroll* association allows a work center to have more than one resource, and a resource is possible to be enrolled in more than one work center.
5. The shifts are abstracted as a *Shift* class with attributes of *resource*, *beginEpk* and *endEpk*. The *Resource* class is a composite of the *Shift* class.

### 3.2. Planning and scheduling model

A demand is a request made by a customer for a quantity of the finished product with a due date specified. The demand arrives on a continuous basis; however, the planning and scheduling of the demand can be processed either continuously or periodically. A production system can either accept all coming demands or reject some of them by a judgment. A planned delivery date is set by a negotiation with the customer.

The planning and scheduling process, in turn, generates requests for the raw materials. The request is in a similar form with the demand that indicates raw material, quantity, and request date.

If the lot-for-lot (L4L) policy is specified, the purchase order, or we say supply from the view point of vender, is just the generated request. Besides the L4L, economic order quantity (EOQ) and periodic order quantity (POQ) rules are usually adopted. The EOQ is to minimize the ordering cost and the inventory carrying cost by assuming a constant demanding rate in the periodic basis. However, Vollman et al. (1997) pointed out EOQ results in a mismatch between supply quantities and actual request quantities in the case of MRP. An improvement of EOQ is POQ that applies EOQ and the demanding rate in computing an economic time between orders (TBO). The requests for the same raw materials within the TBO, are aggregated in their quantities, and arranged to their request dates to form a purchase order. The order has to be released to a vender no later than the procurement lead time before its request date. A planned arrival date for the supply is then determined by a negotiation with the vender.

Components, links, and schedules play an important role in keeping a result of planning and scheduling process under the APPCS. A component is an information object that processes a quantity of parts from a start epoch to a finish epoch. There is a request-provide relationship between two components. The component provides a part for the other components is called a provider, while the component requests a part from other components is a requester. To satisfy a minimum requirement of the APPCS, at least the following types of components are necessary.

1. Demand (DMD): A DMD component is a component form of a demand. It is a pure requester whose requests can be satisfied by INV, SUP, REQ, or WIP component. The start epoch of the DMD component is set to the due date of the demand.
2. Inventory (INV): An INV component is an information object for managing the inventory of a part. It is a pure provider that provides the requests of DMD or REQ component.
3. Supply (SUP): A SUP component is a component form of a purchase order. It is a pure provider that provides requests to DMD or REQ component. The quantity of a SUP component is released at the start epoch and can only be provided after the finish epoch.

4. Net requirement (REQ): A REQ component is generated for filling up a shortage that the gross requirement is unable to be supplemented by INV, SUP, and WIP components. It is both a requester and provider type of component. A REQ component starts processing at the start epoch by requesting necessary parts from its providers. After the REQ component is finished at the finish epoch, it becomes available to its requester.
5. Work-in-process (WIP): While a REQ component is processed, it is possible that its requester is cancelled due to some unpredictable events. The on-processing REQ component can not be cancelled due to a loss of setup time. The REQ component lost its requester becomes a WIP component. The WIP component is a pure provider.

The request-provide relationship of two components is kept as a link. A requester component can ask quantities from one or more provider components. A provider can also provide quantities to several requester components. The link between a provider and a requester is a guarantee of the provider for offering a quantity of part to the requester. A schedule is an actual processing interval for an operation of REQ or WIP component. It is a subinterval of a shift assigned to a resource that is qualified to process the operation. A valid schedule is an interval that has no intersection with other schedules of the same resource.

We model the components, links, and schedules as *Components*, *Links*, and *Schedules* classes respectively. The *Component* class has attributes of *type*, *part*, *quantity*, *startEpk*, and *finishEpk*. The *Link* class, which is also an association whose both ends connect to the *Component* class, has attributes of *requester*, *provider*, and *useQty*. The *useQty* is a promised quantity of a provider to a requester. The *Schedule* class is aggregated to both the *Component* and *Shift* classes with attributes of *component*, *shift*, *operation*, *startEpk*, and *finishEpk*.

The demands are abstracted as a *Demand* class with *part*, *quantity*, *acceptEpk*, *deliveryEpk*, and *demandCom* attributes, and the supplies as a *Supply* class with *part*, *quantity*, *releaseEpk*, *arrivalEpk*, and *supplyCom* attributes. The *acceptEpk* is a promised due date, while the *deliveryEpk*

stores actual date of the delivery. The *demandCom* (*supplyCom*) provides an access link between a demand (a supply) and a component. A class diagram for the planning and scheduling model is shown in figure 5.

[Insert figure 5 about here]

### 3.2.1 Scheduling a component

The scheduling process is to allocate required processing time to the responsible resources. For a REQ component, the scheduling process is to generate schedules for a sequence of operations necessary for producing the part of the component. For each operation, it chooses a resource in the work center that works for the operation, and allocating available working time of the resource to the operation. The scheduling process places schedules backwardly from a future epoch or forwardly from an epoch into the future.

The *backwardScheduling*() method of the *Component* Class shows how a provider component is backwardly scheduled from an epoch that is the latest start epoch of its requester component. In the method, a *getAptResource*() method of the *WorkCenter* class is a rule to pick up a resource among resources in the work center. The alternative rules are (1) latest starting epoch (LSE) and (2) lowest capacity rate (LCR), etc. To schedule an operation backwardly from its planned finish epoch, the LSE rule picks a resource that can start the operation the latest. The LCR chooses a resource that has the lowest capacity rate during a period of time. The *backwardScheduling*() method calls a *getPrevInterval*() method of the *Resource* class to get a free interval of a resource before an epoch. To a resource, a valid free interval is a subset of one of its shifts and has no overlaps with any planned schedules of the resource.

In the following methods, the class name with an italic font, such as *Shift*, means a set of instances that are of the class type, and the class name with a normal font, such as Shift, means the type of the class. The *forwardScheduling*() and *getNextInterval*() methods have a similar structure with the *backwardScheduling*() and *getPrevInterval*() method, we leave the details to the readers.

***Resource.getPrevInterval (endEpk: Epoch): Interval***

```
Shift F:= {f|f∈Shift & f.resource=this & f.beginEpk<endEpk };
Schedule H:= { h | h∈Schedule & h.shift.resource=this & h.startEpk<endEpk };
Interval i:= NEW Interval(0, 0);
i.endEpk:=MAX { t | (∀f∈F) (t∈f) & (∀h∈H) (t∉h) & t≤endEpk};
i.beginEpk:=MAX {f.beginEpk | f∈F & f.beginEpk≤i.endEpk≤f.endEpk};
H':= { h | h∈H & i.beginEpk≤h.finishEpk≤i.endEpk};
IF H' ≠ { }
    i.beginEpk:=MAX { h.finishEpk | h∈H' }};
RETURN i;
```

***Component.backwardScheduling ( ): void***

```
Epoch latestFinishEpk:=MIN {l.requester.startEpk | l∈Link & l.provider=this};
Operation O:= {o | o∈Operation & o.part=this.part};
IF O={ }
    this.finishEpk:= latestFinishEpk;
    this.startEpk:=this.finishEpk−this.part.leadTime;
ELSE
    FOR o∈O DESCENDING BY o.oprId
        Time reqTime:=o.setupTime+o.processTime × this.quantity;
        Resource r=: o.workCenter.getAptResource( latestFinishEpk, reqTime)
        WHILE prcTime>0
            Interval i:=r.getPrevInterval( latestFinishEpk);
            reqTime:=reqTime−( i.endEpk−i.beginEpk);
            IF prcTime<0
                i.beginEpk:=i.beginEpk−prcTime;
            Schedule:=Schedule∪ {NEW Schedule(this, o, r, i.beginEpk, i.endEpk)};
            latestFinishEpk:=i.beginEpk;
        ENDWHILE
    ENDFOR
    this.startEpk:= latestFinishEpk;
    this.finishEpk:=MAX{ h.finishEpk | h∈Schedule & h.component=this};
ENDIF
RETURN
```

### 3.2.2 Planning a component

The planning process of a component is to determine the provider components for the requester component. For a requester component, the planning process grants work-in-process, released supply, and inventory to its grass requirements by making links to WIP, SUP and INV components. The unsatisfied quantity, the net requirement, is then filled up by a new REQ component. The *planning*( ) method of the *Component* class describes how a REQ component is planned. If start epoch of the component is not yet decided, in the case of forward scheduling a demand, the parameter *t* gives an epoch for determining whether WIP, SUP can be used by the component or not.

Figure 6 shows how component A is backwardly scheduled to 3 schedules and planed to 3 provider components. In the result of scheduling, resource **P** is chosen for that it can start later than **Q** for operation **10**. In the result of planning, a supply for **B** arrives too late **(50t)** for providing itself to component A.

```
Component.planning ( t: Epoch )
Epoch ifCanUseEpk:=this.startEpk;
IF ifCanUseEpk=null
    ifCanUseEpk:=t;
Bom B=: {b|b∈Bom & b.part=this.part};
IF this.type='DMD'
    Demand {d}:={d|d∈Demand & d.demandCom=this};
    B:={NEW BOM(null, d.part, 1)};
ENDIF
FOR next b∈B
    float reqQty:=b.usage×this.quantity;
    Component C:={c|c∈Component & c.part=b.childPart};
    Component W:={c|c∈C & c.type='WIP' & c.finishEpk≤ifCanUseEpk};
    Component S:={c|c∈C & c.type='SUP' & c.finishEpk≤ifCanUseEpk};
    Component I:={c|c∈C & c.type='INV'};
    FOR c∈(W∪S∪I) ORDER BY c.type='WIP', 'SUP', 'INV' ASENDING BY c.finishEpk
        float netQty:=c.quantity−∑{l.useQty|l∈Link & l.provider=c};
        reqQty:=reqQty−netQty;
        IF reqQty≤0
            Link:=Link∪{NEW Link(this, c, reqQty+netQty)};      BREAK;
        ELSE
            Link:=Link∪{NEW Link( this, c, netQty)};
        ENDIF
    ENDFOR
    IF reqQty>0
        Component:=Component ∪ {NEW Component("REQ", b.childPart, reqQty, null, null)};
        Link:=Link∪{NEW Link(this, c, reqQty)};
    ENDIF
ENDFOR
RETURN
```

[Insert figure 6 about here]

### 3.2.3 Planning and scheduling a demand

The planning and scheduling process is to convert a demand to a set of components, links, and schedules under inventories, released supplies, and work-in-processes. The process is a sequence of a combination of planning processes and scheduling processes.

Planning and scheduling of a demand in a top-down sequence from the DMD component to the REQ components of raw material is called backward scheduling of the demand. The demand is backward scheduled from its promised delivery epoch. If the start epoch of any component of the demand locates in the past, then the final schedules are infeasible and a forward scheduling approach is substituted to plan and schedule the demand from the present time.

Forward scheduling first plans components of a network in a top-down sequence, and then schedules the components in a bottom-up sequence. The processing and scheduling process continues until all the REQ components are planned and scheduled. A sequence of planning and/or scheduling components in a network is determined by the following rules.

1. If backward scheduling is applied, no component can be planned before it is scheduled. A component is in a ready-for-schedule state of backward scheduling if its requester component is planned. A component is in a ready-for-plan state of backward scheduling if it is scheduled.
2. If forward scheduling is applied, no component can be scheduled before it is planned. A component is in a ready-for-plan state of forward scheduling if its requester component is planned. A component is in a ready-for-schedule state of forward scheduling if it has no provider or its provider components are scheduled.
3. If more than two components are in ready-for-schedule, or ready-for-plan state, then other priority rules are necessary to determine their scheduling or planning sequence.

The sequence represents a priority for a component to consume quantities of valid provider components and capacities of valid resources. We demonstrate two methods of *Demand* class, *backwardScheduling*() and *forwardScheduling*(), in the following context to solve the problem of backward scheduling and forward scheduling. In both methods, *getNextScheduleComponent*() or *getNextPlanComponent*() is a rule to determine the next component for scheduling or planning.

***Demand.backwardScheduling (): Component***

Component *dc*:=NEW Component("DMD", *null*, *null*, *this.deliveryEpk*, *null*);

```
Component := Component∪{dc};
Component pList:={ dc}, sList:={dc}, dList:={};
WHILE pList≠{}
    THREAD
        Component nsc:=getNextScheduleComponent(sList);
        dList:=dList∪{nsc};
        nsc.backwardScheduling();
        sList:=sList−{nsc};
        pList:=pList∪{nsc};
    ENDTHREAD
    THREAD
        Component npc:=getNextPlanComponent(pList);
        npc.planning(null);
        pList:=pList−{npc};
        sList:=sList∪{l.provider|l∈Link&l.requester=npc&l.provider.type="REQ"};
    ENDTHREAD
ENDWHILE
RETURN dList;
```

***Demand.forwardScheduling ( t: Epoch ): void***

```
Component dc:=NEW Component("DMD", null, null, this.deliveryEpk, null);
Component := Component∪{dc};
Component pList:={dc}, sList:={dc};
WHILE PlanList≠{}
    Component npc := getNextPlanComponent(pList);
    sList:=sList∪{npc};
    npc.planning(t);
    pList:=(pList∪{l.provider|l∈Link&l.requester=npc})−{npc};
ENDWHILE
WHILE sList≠{}
    Component nsc := getNextScheduleComponent(sList);
    nsc.forwardScheduling(t);
    sList:=sList−{nsc};
ENDWHILE
RETURN;
```

In the case of backward scheduling, the scheduling process decides member of planning list, the planning process decides member of scheduling list, and both processes runs concurrently, hence a sequence can not be uniquely identified by the scheduling rule and the planning rule only. To uniquely identify a sequence, a further constrain is necessary. For example, a breadth-first sequence can be achieved by the constraint that the planning process can not start until the schedule list is empty and the scheduling process can not start until the plan list is empty. A depth-first sequence can be organized by assuming that schedule list is a stock and a component is planned immediately after it is scheduled.

In the case of forward scheduling, the planning process appends provider components to the planning list and scheduling list, and use a planning rule to choose the next component for planning. After planning is finished, the scheduling process chooses the next component for scheduling among the scheduling list by a scheduling rule. The sequence is possible to be uniquely identified by using only a planning rule and a scheduling rule.

Figure 7 shows 3 sequences of planning and scheduling by assuming depth-first rule, breadth-first rule, and no rule. In the figure, *S* indicates scheduling list, *P* planning list, *Sch*(*x*) scheduling component *x*, and *Plan*(*y*) planning component *y*.

After all demands are planned and scheduled, a set of REQ components that have no provider component is generated for requesting a quantity of raw materials. The L4L, EOQ, or POQ rules are applied, and safety buffers are considered to generate a set of SUP components instead of the REQ components. If a requester component whose net requirement is supplied by a SUP component, then a link from the SUP component to the requester component with a request quantity is created.

[Insert figure 7 about here]

### 3.3. Control model

The production control of APPCS is a process to manage and monitor the execution of planned components and their schedules under uncertainties. An uncertainty is an unpredictable event that could make the production plan invalid. If the production system does not learn to deal with it, it is prone to cause the production system an inconceivable loss. For example, a demand change caused by customer, forecasting error, and dramatic changes of cost in the market are such events.

APPCS response to the events by redoing planning and scheduling. An event will cause one or more component of the network unattainable. Except for the on-processing component, the components of the demand whose direct or indirect provider component is unattainable will be cancelled and the processing and scheduling is run again for the demand. The control process

continues for all the demands.

[Insert figure 8 about here]

The result of planning and scheduling of all demands becomes a network composed of components and links. A network has a state, and an event brings about state change to the network. Figure 8 depicts how state ($S_n$) of a network is changed by an event ($E_n$). In the figure, a dot line with its start point locates in the time axis and end arrow pointing to a component shows an event, and a bold line indicates component or link that has a direct or indirect request-provide relationships with the component. Components affected by the event are possible to be unattainable.

Whybark and Williams (1976) pointed out sources of uncertainty are (1) supply timing, (2) supply quantity, (3) demand timing, and (4) demand quantity. We model the events as an *Event* class with its specific classes as shown in figure 9. The *notifyEpk* attribute of the class keeps the value of epoch that the production system knows the event. For example, a supply change event can be notified in advance by its vender before the promised delivery epoch. The specific classes are corresponding to individual types of event described in the following.

[Insert figure 9 about here]

1. After a purchase order is released for a supply, the event of *SupplyUncertainty* type is caused by the vender notifying its delay in delivering and/or shortage of the supply. Any accident occurred during transportation is also a possible reason for the uncertainty. A new delivery epoch and/or a delivery quantity should also be provided by the vender.
2. After planning and scheduling is done for a demand, the event of *DemandUncertainty* type is caused by the customer notifying its change in demand quantity and/or promised delivery epoch. A correct request quantity and/or a new negotiated delivery epoch should be followed by the event.

Once an event is notified, *handle*() method of the *Event* class is triggered to handle the event. To run the method, a *status* attribute belongs to the *Component* class is added to show the status of a component whose possible values is "INITIAL" or "CHANGED". A component is initialized to be "INITIAL" after it is planned and scheduled. The components that are directly affected by an event will be set to "CHANGED".

In the *handle*() method, the changed objects are updated, directly related components are set to be "CHANGED", a set of demands that at least one of its provider components is in "CHNAGED" status is searched by a static *getAffectedDemand*() method, the not yet started components, which are either REQ or SUP, that are planned for the demands are cancelled by a *cancelInvalidPlan*() method of the *Demand* class, and finally the demands are planned and scheduled again according to a priority of the demands. The priority, determined by a static method *getNextDemand*(), is possible the earliest due date (EDD), or the shortest processing time (SPT) rule. Each demand is first planned and scheduled by *backwardScheduling*() method. If the result of the backward scheduling is invalid, then *forwardScheduling*() is adopted instead.

***Event.handle (): void***

```
CASE this.type ="DemandUncertainty"
    this.demand.quantity:=this.changedQty;
    this.demand.deliveryEpk:=this.changedDeliveryEpk;
    this.demand.demandCom.status:="CHANGED";
CASE this.type ="SupplyUncertainty"
    this.supply.quantity:=this.changedQty;
    this.supply.arrivalEpk:=this.changedArrivalEpk;
    this.supply.supplyCom.status:="CHANGED";
ENDCASE
Demand D:=getAffectedDemand();
FOR Demand d∈D
    d.cancelInvalidPlan(this.notifyEpk);
WHILE (Demand d:=getNextDemand(D))≠{}
    Component C:=d.backwardScheduling();
    IF (∃c∈C).c.startEpk ≤ this.notifyEpk
        d.cancelInvalidPlan(this.notifyEpk);
        d.forwardScheduling();
    ENDIF
    D:=D−{d};
ENDWHILE
this.delete();
```

```
RETURN

Static getAffectedDemand (): Demand
Component cList:={c|c∈Component&c.status="CHANGED"}
Component dList:={}
FOR c∈cList
    IF c.type = "DMD"
        dList:=dList ∪ {c};
    cList:=cList ∪ {l.requester|l∈Link&l.provider=c} − {c};
ENDFOR
RETURN {d|d∈Demand&d.demandCom ∈dList}]

Demand.cancelInvalidPlan (notifyEpk: Epoch): void
Component pList:={this.demandCom};
ENDFOR
FOR c∈pList
    L:={l|l∈Link&l.requester=c};
    RC:={l.provider|l∈L&l.provider.type="REQ" & l.provider.startEpk>notifyEpk};
    SC:={l.provider|l∈L&l.provider.type="SUP" & l.provider.startEpk>notifyEpk};
    pList:=(pList ∪RC∪SC)−{c};
    Schedule:=Schedule − {h|h∈Schedule&h.component ∈RC};
    Link:=Link−L;
ENDFOR
RETRUN
```

## 4. An APPCS based simulator integrated with APS

For calculating necessary rate of safety lead time buffer and/or safety stock against a rate of supply time or quantity uncertainty to attain a service level under APPCS, a simulator is designed by following the APPCS model and implemented by integrating a scheduler of advanced planning and scheduling (APS) that is provided by Symix (1998).

### 4.1. Simulation design

The data for simulation is drawn from an ABC plant. Sets of all parts, operations, work centers, resources, shifts, demands, and supplies in the ABC plant are defined as *Part*, *Operation*, *WorkCenter*, *Resource*, *Shift*, *Demand*, and *Supply* respectively. A set of links between the higher and the lower level of parts is *Bom*.

Assume that all of the products have respective three-level BOM. The set of part in level $i$ is $P_i$, $1 \leq i \leq 3$. The number of $P_1$ is 8, which is expressed by $|P_1|=8$, and $|P_2|=12$, $|P_3|=20$. For a part $p \in P_i$, a number of component parts $B(p)=\{b.childPart \mid b \in Bom \,\&\, b.part=p\}$ are randomly

selected from $P_{i+1}$, and the number satisfies $1 \le |B(p)| \le 4$. For a $b \in Bom$, its usage satisfies $1 \le b.usage \le 4$. For a part $p \in P_i$, $1 \le i \le 2$, a set of operations $O(p) = \{o \mid o \in Operation \,\&\, o.part = p\}$ is generated by following $1 \le |O(p)| \le 6$. For a raw material $p \in P_3$, the purchase lead time satisfies $0\,\mathrm{h} \le p.leadTime \le 48\,\mathrm{h}$.

For an operation *o*, *o.workCenter* is randomly selected from *WorkCenter*, its setup time satisfies $0\,\mathrm{h} \le o.setupTime \le 0.5\,\mathrm{h}$, and processing time per item $0\,\mathrm{h} < o.processTime \le 0.2\,\mathrm{h}$. The number of work centers is 4, and each work center has a distinct resource. In the simulation every resource has the same shift that works from 0:00 to 10:00 everyday without vacation or maintenance. In the simulation, above product data are randomly sampled from respective uniform distributions with the ranges. The planning horizon is defined within 31 days from 2000/01/01 00:00 to 2000/01/31/23:59. The required quantity for a demand $d \in Demand$ varies according to the exponential distribution with the average of 10 pieces. The following rules are used in the simulation for planning and scheduling.

1. As planning and scheduling a set of demands, we adopt EDD rule by granting priority to the demand that has the earliest due date, and design it in the *getNextDemand*() method.
2. To uniquely identify a sequence of planning process and scheduling process in backward scheduling of a demand, the "depth-first sequence rule" is specified in the simulation.
3. The next component selected for scheduling is the component that has the earliest finish epoch as implemented in *getNextScheduleComponent*().
4. The next component select for planning is the component with the earliest start epoch as shown in *getNextPlanComponent* ().
5. The LSE rule is used in selecting an apt resource among resources in a work center. The rule is implemented in the *getAptResource*() method of the *WorkCenter* class.

***Static getNextDemand ( D: Demand ): void***
Demand $\{d\}$ :=MAX$\{d.acceptEpk \mid d \in D\}$
RETURN *d*;

***Static getNextScheduleComponent ( C: Component ): void***

Component $\{c\}$ :=MAX$\{ c.finishEpk | c \in C \}$
RETURN *c*;

***Static getNextPlanComponent ( C: Component ): void***

Component $\{c\}$ :=MAX$\{ c.startEpk | c \in C \}$
RETURN *c*;

***WorkCenter.getAptResource ( latestFinish: Epoch, PrcTime: Time ): Resource***

```
Resource aptRes:=null;
Epoch latestStartEpoch:=0;
FOR r∈ { e.resource | e∈Enroll & e.workCenter=this }
    Interval i;
    Time reqTime:=prcTime;
    Epoch lfe:=latestFinish;
    WHILE reqTime>0
        i:=r.getPrevInterval(lfe);
        reqTime:=reqTime−(i.endEpk−i.beginEpk);
        IF reqTime<0
            i.beginEpk=i.beginEpk−reqTime;
        lfe:=i.beginEpk;
    ENDWHILE
    IF i.beginEpk>latestStartEpoch
        latestStartEpoch:=i.beginEpk;
        aptRes:=r;
    ENDIF
ENDFOR
RETURN aptRes;
```

We adopt one periodic order quantity $(\mathrm{POQ}=1)$ rule to decide the lot size of procurement. The *procurement*() method shows how set of SUP components is generated from the result of planning and scheduling in the simulation. The specification of safety buffers makes a SUP component have an earlier *startEpk* or request a more *quantity* than required.

***Static procurement (bufferType, bufferRate: float ): void***

```
FOR p∈P3
    Component C:={ c | c∈Component & c.type="REQ" & c.part=p };
    IF C≠{ }
        Epoch finishEpk:=MIN{ c.finishEpk | c∈C };
        float quantity:=∑{ c.quantity | c∈C };
        Epoch startEpk:=fnishEpk−p.leadTime;
        IF bufferType="SafetyStock"
            quantity:=quantity×(1+bufferRate);
        IF bufferType="SafetyLeadTime"
            startEpk:=startEpk−bufferRate×p.leadTime;
        Component nc:=NEW Component("SUP", p, quantity, startEpk, finishEpk);
        FOR l∈{ l | l∈Link & l.provider ∈ Com}
```

```
        l.provider:=nc;
    Component:=Component ∪ {nc} − C;
  ENDIF
ENDFOR
RETURN
```

Now we proceed to uncertainty-related issues. For a purchase order $s \in Supply$, its notification time is located between *s.releaseEpk* and *s.arrivalEpk*. The shortage of supply quantity is smaller than *s.quantity*. The delay of supply is smaller than *s.part.leadTime*.

For a demand $d \in Demand$, notification of change occurs before the earliest start epoch of components that are related to *d*. The time when a customer asks earlier shipment than initial due date is smaller than the total processing time of the demand. Extra quantity requested by a customer for an initial demand is less than the demand's requirement quantity *d.quantity*. All of these uncertainties are sampled from suitable uniform distributions in simulation.

A test case in simulation is a combination of the following five parameters.

1. Advanced notification is possible or not.
2. Source of uncertainty specifies either demand uncertainty or supply uncertainty.
3. Type of uncertainty specifies either time or quantity.
4. Level of uncertainty. Degree of demand uncertainty is high (75%), medium (50%) or low (25%). The degree is defined as the ratio of the changed demands to the total demands. Degree of supply uncertainty is high (50%), medium (25%) or low (12.5%), which is defined similarly.
5. Rate of buffer. Degree of safety lead time buffer takes a value in {0, 0.1, 0.2, ..., 1.0}. It is defined as the rate of additionally reserved lead time to the predefined procurement lead time. Degree of safety stock buffer is a rate of additional quantity to total required quantity.

[Insert figure 10 about here]

For example, a case has high-level uncertainty in supply quantity and 20% safety lead time with advanced notification. Then, APPCS schedules with procurement lead times 1.2 times longer

than usual, and 75% of purchase orders will change in respective quantities. Such a change is notified from a supplier and causes rescheduling, depending on the epoch of notification. Each case is simulated 40 times to provide a data for the case, and performance indices are calculated for the data.

Figure 10 shows a class diagram for the simulation design. A test case is set by specifying values of *repeatTimes*, *eventType*, *eventRate*, *eventLevel*, *bufferType*, *bufferRate*, and *ifNotify*. Based on specification of the test case, values of *Part*, *Bom*, *Operation*, *WorkCenter*, *Resource*, *Shift*, *Event*, *Demand*, *Supply*, *Component*, *Link*, and *Schedule* are initialed for a simulation by *initialize*() method. Then the simulation is run by the *run*() method, and finally its result is saved for further analysis by the *save*() method. A design spec. of the *run*() method is demonstrated as follows.

```
Simulation.run (): void
this.initialize();
WHILE (Demand d:=getNextDemand(Demand)≠{})
    d.backwardScheduling();
ENDWHILE
procurement(tc.bufferType, tc.bufferRate);
FOR e ∈ Event ASENDING BY e.notifyEpk
    e.handle();
procurement(tc.bufferType, tc.bufferRate);
ENDFOR
this.save();
RETURN
```

### 4.2. Simulation result

The simulation result is analyzed and evaluated by service level. The service level is defined as an average ratio between the number of demands delivered in time and the number of all demands as shown in (1).

$$\textit{service level} = \frac{|\{d \mid d \in Demand \ \& \ d.deliveryEpk \le d.acceptEpk\}|}{|Demand|} \quad (1)$$

Figure 11a shows the relationship between service level and rate of buffering. The hollow curves depict that relationship with safety stock (sb_), while the solid curves depict that with safety lead time (lb_). For both kinds of lines there are three cases. They are under high (_htu), medium

(_mtu) and low (_ltu) levels of supply time uncertainty. The higher the rate of uncertainty is, the lower the service level can be achieved. We observe that the service level increases much sharply with rate of safety lead time buffer than that of safety stock buffer.

The reason why safety time is effective against supply time uncertainty is two fold. When the prescribed safety time is longer than delay time, or when the notification time is early enough, the uncertainty won't delay the production. This result suggests that safety lead time can be adjusted by the APPCS to achieve a target level of service in different degree of uncertainty environments. When safety stock is used to against supply time uncertainty, there is an upper limit of service level for each level of uncertainty. Hence, safety time is more flexible than safety stock under supply time uncertainty.

Figure 11b shows the relationship between service level and supply quantity uncertainty. Since each service level goes gradually up to 100% as rate of both buffers increased, both types of buffer are effective against supply quantity uncertainty. When supply quantity uncertainty is informed, some action must be taken to supplement the shortage. If safety time is reserved, releasing another purchase order is inevitable and no production can be started until the arrival of the purchase material. If safety stock is reserved, the shortage is supplemented either from stock or by releasing another purchase order. The result of the experiment shows that both buffering approaches are adjustable to achieve a target service level under supply quantity uncertainty.

Figure 11c and 11d shows the result of the same experiment with figure 11a and 11b respectively, but the plant won't know the change until the promised delivery date. Thus, the effect of notification on service level can be obtained by the difference of the two pairs of figures. By comparing figures 11a with 11c, we observe that safety time is more effective than safety stock. The comparison of figure 11b and 11d brings us the fact that the service level for using safety lead time falls dramatically.

[Insert figure 11 about here]

Among the related works in comparing safety time and safety stock, the pioneer research is done by Whybark and Williams (1976) who compares the two buffers under both demand and supply uncertainty. They modeled a representative part in an MRP system. They concluded that safety time buffer can protect from time uncertainty and safety stock buffer performs well under quantity uncertainty in either case of demand or supply. Grasso and Taylor (1984) and Buzacott and Shanthikumar (1994) have done a related research and have different suggestions. Hegedous and Hopp (2001) reveal that the setting of safety lead time provides production flexibility in the form of build ahead that helps deal with capacity issues.

Under an assumption that any uncertainty will be informed by demand or supply side earlier than shipment or delivery date, the safety time acts more flexible in response to the four types of uncertainty than the safety stock. Besides, we found that the earlier notification policy, which enables the releasing of another purchase order if the rescheduling suggests, shifts the service level and decreases the delay time dramatically under demand side uncertainties.

We conclude that the approach of earlier notification plays an important role in supporting policy of safety time buffering both to achieve a higher service level under supply time uncertainty and to keep the same service level with safety stock under supply quantity uncertainty. The notification approach can be regarded as another form of safety time that is offered by external venders and with less effort.

## 5. Conclusion

We abstract agile production planning and control system as the APPCS model that contains static model, planning and scheduling model, and control model by using UML. The objects in static model are similar to usual MRP, consisting part, BOM, operation, work center, resource, and shift (work hours). The planning and scheduling model provides a flexible structure for planning and scheduling, and production control. The control model defines uncertainties and their impacts

on the other objects. The structure is a network which consists of components and links. A component is a requester, provider, or both. A link represents that a requester consumes quantities of a provider. Planning and scheduling process changes the state of a network.

If some priority rules and parameters, such as lot size, lead time, safety stock, are specified, then a feasible production plan can be generated and regenerated against uncertainties. The APPCS model has been shown useful by applying it in an implementation of a simulator.

The simulator has been used to advise a necessary rate of safety lead time buffer and/or safety stock against a rate of supply time or quantity uncertainty to attain a service level. It showed that when notification system is set, safety lead time buffer performs well under both time and quantity uncertainty, but safety stock buffer is only effective in against quantity uncertainty. The approach of earlier notification is a complementary policy for safety time approach under supply uncertainties.

In order to control a business process with high quality of performance, qualitative analysis of dynamic property such as Sato (1999) is not sufficient. The design of dynamics of a business process is necessary. If we could bring planning components into the design of business processes, then the whole control mechanism can be explicitly managed. The result of this paper plays a basic role for that purpose.

**Acknowledgement**

This research was partially supported by the Ministry of Education, Science, Sports and Culture of Japan, Grant-in-Aid for Scientific Research (C) 13630132, and by the University of Tsukuba, Special Research Grant (S). The authors are grateful to SAP Japan™ and FrontStep Japan™ (formerly, SIMIX Japan) for their support and help.

**Reference**

[1] A.W. Scheer, 1994, Business Process Engineering, 2nd edition, Springer.

[2] D.C. Whybark, J.G. Williams, 1976, Material requirements planning under uncertainty. Decision Sciences 7, 595-606.

[3] D.E. Shobrys, D.C. White, 2002, Planning scheduling, and control systems: why cannot they work together, Computers and Chemical Engineering 26, 149-160.

[4] E.A. Silver, D. F. Pyke, R. Peterson, 1998, Inventory Management and Production Planning and Scheduling, 3rd edition, John Wiley & Sons.

[5] E.M. Goldratt, J. Cox, 1986, The Goal, Groton-on-Hudson, New York, North River Press.

[6] E.T. Grasso, B.W. Taylor, 1984, A simulation-based experimental investigation of supply/timing uncertainty in MRP systems, International Journal of Production Research 22, 485-497.

[7] J.A. Buzacott, J.G. Shanthikumar, 1994, Safety Stock versus Safety Time in MRP Controlled Production Systems, Management Science 40, 1678-1689.

[8] L.M. Sanchez, R. Nagi, 2001, A review of agile manufacturing system, International Journal of Production Research 39 16, 3561-3600.

[9] M.G. Hegedus, W.J. Hopp, 2001, Setting procurement safety lead-times for assembly systems. International Journal of Production Research 39, 3459-3478.

[10] OMG, 2002, OMG Unified Modeling Language Specification (Action Semantics)-V1.4.

[11] R. Sato, T.L. Tsai, 2002, An agile production planning and control with additional purchase orders, Submitted.

[12] R. Sato, H. Praehofer, 1997, A discrete event model of business system- A Systems Theoretic Foundation for Information Systems Analysis: Part 1, IEEE Transactions on Systems, Man, and Cybernetics 27, 1-10.

[13] R. Sato, 1997, Meaning of Dataflow Diagram and Entity Life History – A Systems Theoretic Foundation for Information Systems Analysis: Part 2, IEEE Transactions on Systems, Man, and Cybernetics 27, 11-22.

[14] Symix Systems Inc., 1998, Order Links Manual.

[15] T.E. Vollman, W.L. Berry, D.C. Whybark, 1997, Manufacturing Planning & Control Systems, New York, McGraw-Hill.

[16] Y. Tu, 1997, Production planning and control in a virtual One-of-a-Kind Production company, Computers in Industry 34, 271-283.

**All Figure Captions**

Figure 1. An example of constructs in the Core package

Figure 2. A product structure diagram of a snow shovel (Vollman et al., 1997)

Figure 3. An example of operations, work centers, resources, and shifts

Figure 4. The static model

Figure 5. The planning and scheduling model

Figure 6. A result of planning and scheduling a component

Figure 7. Sequences of planning and scheduling

Figure 8. State change of a network by events

Figure 9. The control model

Figure 10: The simulation model

Figure 11. Service level under supply time/quantity uncertainty

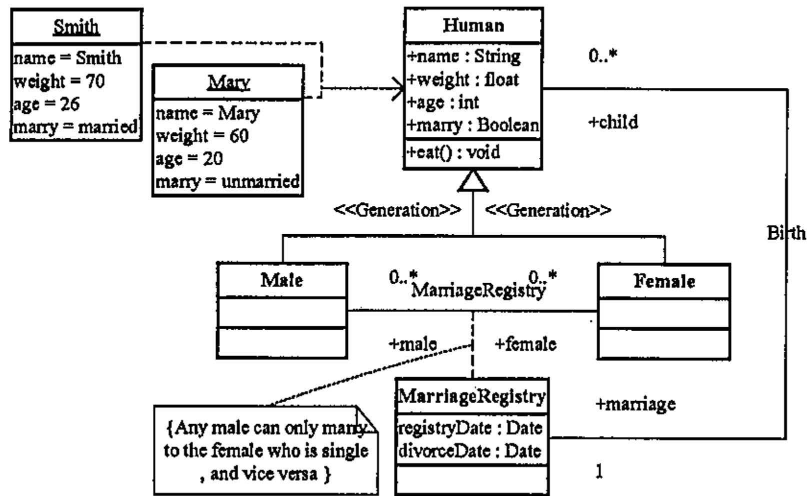



Figure 1. An example of constructs in the Core package

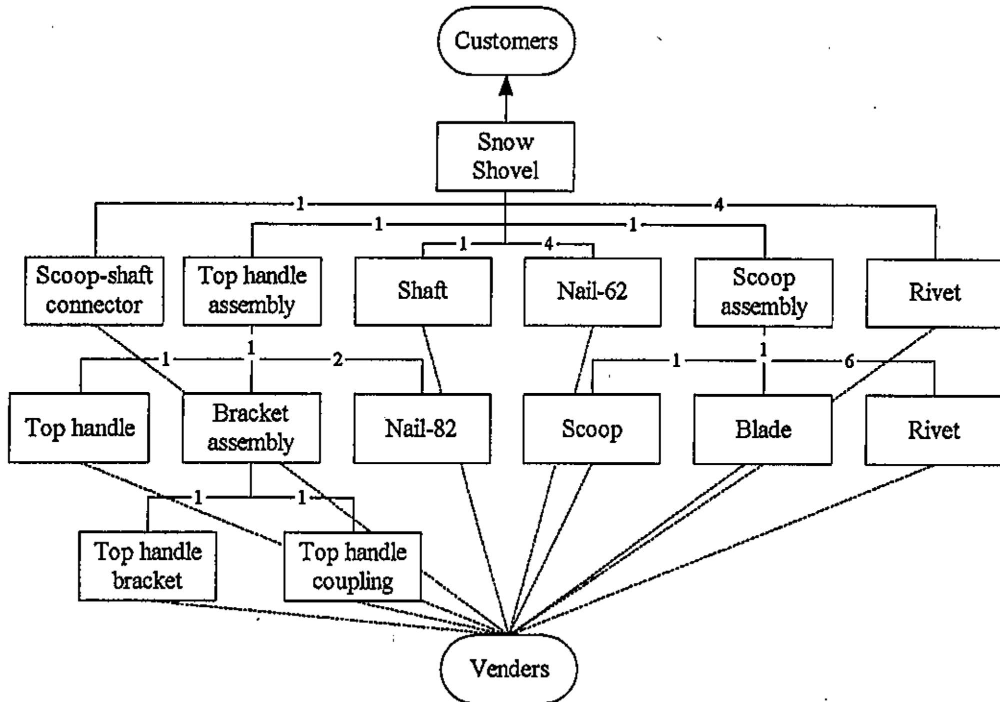



Figure 2. A product structure diagram of a snow shovel (Vollman et al., 1997)

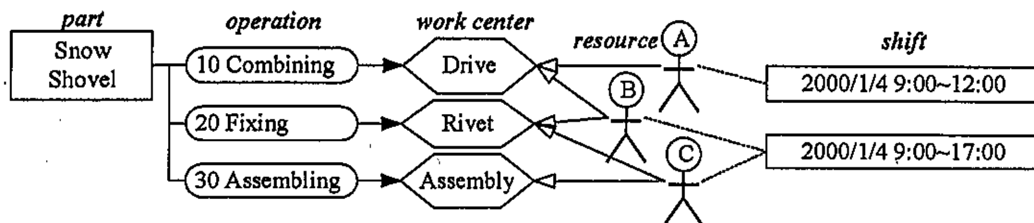



Figure 3: An example of operations, work centers, resources, and shifts

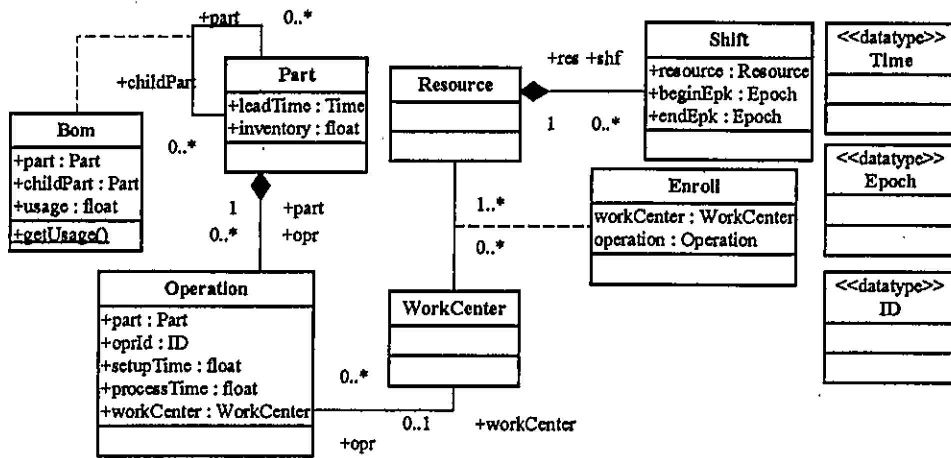



Figure 4: The static model

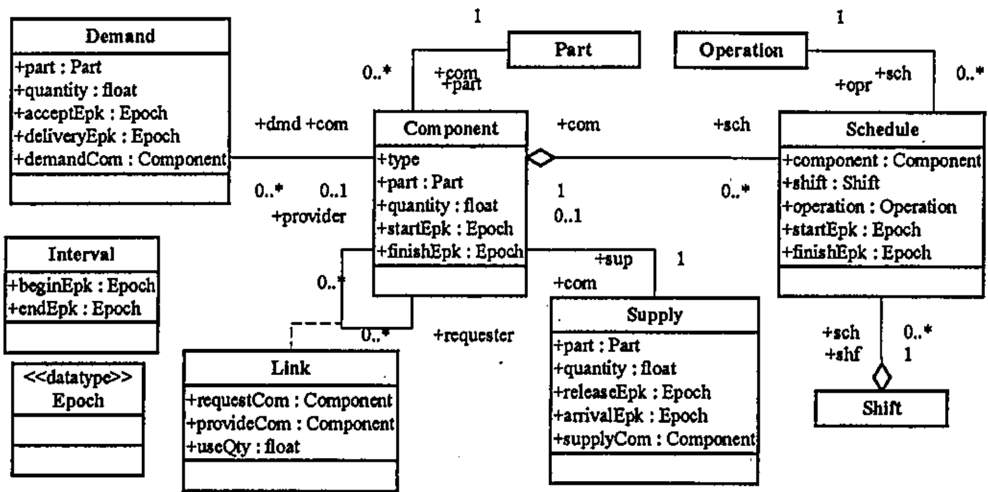



Figure 5. The planning and scheduling model

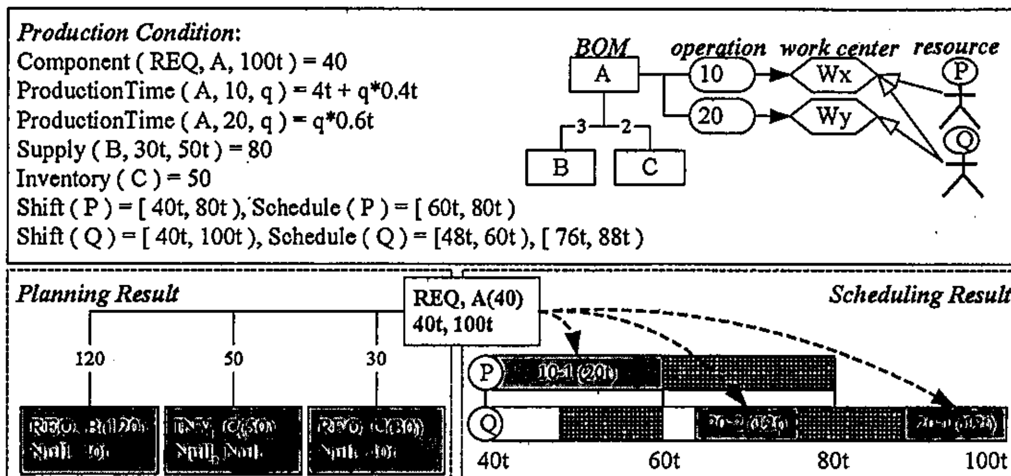



Figure 6. A result of planning and scheduling a component

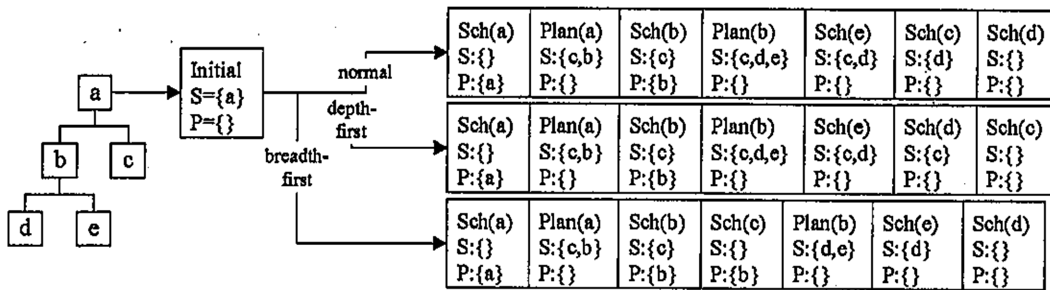



Figure 7. Sequences of planning and scheduling

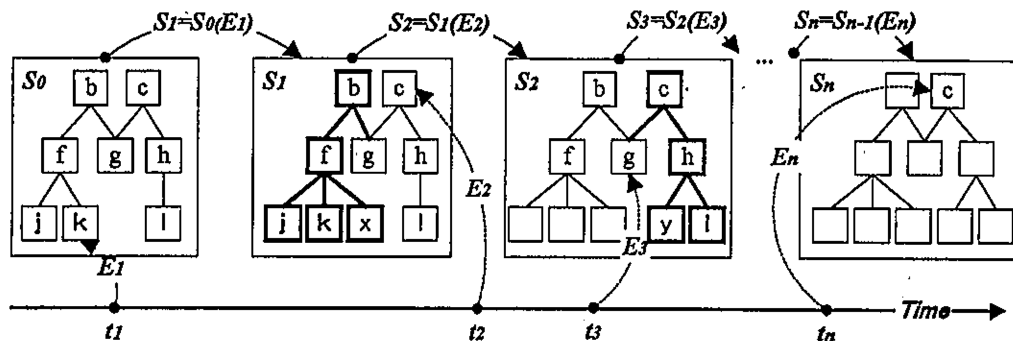



Figure 8: State change of a network by events

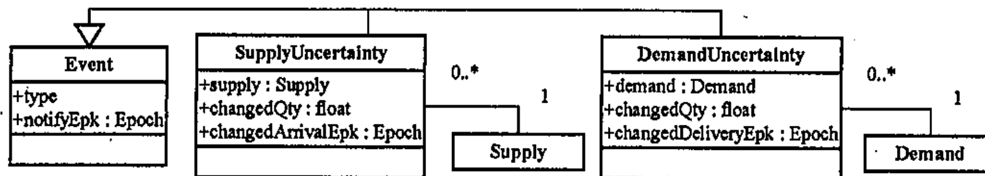



Figure 9: The control model

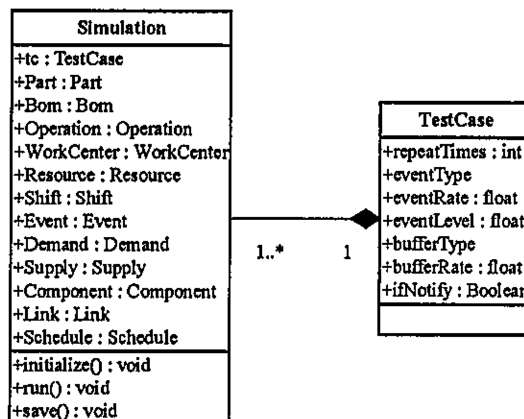



Figure 10: The simulation model

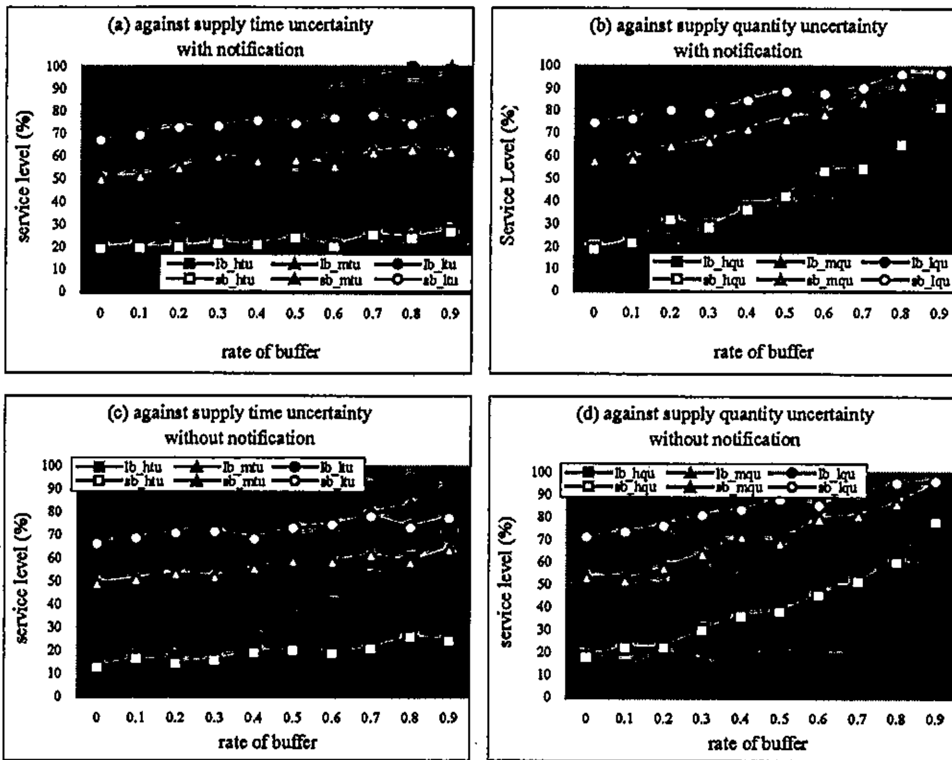



Figure 11. Service level under supply time/quantity uncertainty