Artificial Intelligence for Engineering Design, Analysis and Manufacturing (1998), *12*, 29–42. Printed in the USA.

Modelling business processes and enterprise activities at the knowledge level

ROBERT DE SOUZA, ZHAO ZHEN YING, AND LIU CHAO YANG

Centre for Engineering and Technology Management, School of Mechanical & Production Engineering, Nanyang Technological University, Nanyang Avenue, Singapore 639798

(RECEIVED February 28, 1997; REVISED August 20, 1997; ACCEPTED October 3, 1997)

Abstract

As manufacturing systems become more sophisticated and complicated, effective managers know how they play a crucial role in managing an enterprise and managers know how to deal with their dynamics and uncertainty. In this article, a formalism based on the computer-integrated manufacturing open-system architecture (CIMOSA) reference model is presented to specify the business processes and enterprise activities at the knowledge level. The formalism uses an integration of multiple types of knowledge, including precise, muddy, and random symbolic and numerical knowledge to systematically represent enterprise behavior and functionality. To support the modelling process, a prototype is developed and an example for a maintenance activity is presented to demonstrate the effectiveness of the proposed method.

Keywords: Enterprise Modelling; Knowledge Integration; Fuzzy Logic

1. INTRODUCTION

The design and improvement of modern manufacturing enterprises is an extremely complex process. It involves a nontrivial combination of technological, human, machine, and organizational issues. To cope with such a challenging problem, there requires a model of the enterprise not only at shop floor level but also at an organizational and business level. Manipulating the model (such as a simulation) to produce guidelines, decisions, suggestions, or knowledge is advantageous to the design and improvement of the enterprise.

There is a vast body of well-developed enterprise models in the context of computer-integrated manufacturing (CIM), which are ready to be used either in the design or in the implementation phases of enterprise integration. For instance, the International Standard Organization (ISO) Technical Committee TC 184 on enterprise modelling and integration has proposed a reference model for shop floor production standards (ISO, 1990). This model provides a conceptual framework for understanding discrete parts manufacturing; and can be used to identify areas of standards necessary to integrate manufacturing systems. The Purdue Consortium has developed an engineering-oriented architecture and associated implementation methodology. The architecture was published as the Purdue Enterprise Reference Architecture (PERA) (Williams, 1994; Rathwell & Williams, 1995). The Graphés à Résultatats et Activités Interreliés (GRAL) laboratory of the University of Bordeaux has also developed a framework and modelling tools to support enterprise integration (Doumeingts et al., 1987). ESPRIT projects have also developed modelling reference architectures and industrial applications. The most important results have been the computer-integrated manufacturing open system architecture (CIMOSA). In the CIMOSA scenario, the business process is distinguished from enterprise activities. An enterprise is viewed as a large collection of concurrent business processes executed by a set of functional entities that contribute to business objectives or goals (Vernadat, 1995, 1996). The Architecture for integrated Information System (ARIS) has been developed at the University of Saarbrüchen in Germany (Scheer & Kruse, 1994). Its overall structure is similar to CIMOSA, but it deals with more traditional business-oriented issues of enterprises such as order processing, inventory control, etc. To combine the various methodologies and modelling techniques, Bernus and Nemes (1995) proposed a new Generic

Reprint requests to: Robert de Souza, Centre for Engineering and Technology Management, School of Mechanical & Production Engineering, Nanyang Technological University, Nanyang Avenue, Singapore 639798. E-mail: mrdesouza@ntu.edu.sg

Enterprise Reference Architecture and Methodology (GERAM), which defines a toolbox of concepts for designing and maintaining enterprises during their entire life cycle. GERAM attempts to predict system behavior and gain insight into interaction between the three primary functional components of an enterprise: manufacturing, marketing, and R&D, and the interaction of the enterprise with other enterprises, Mujataba (1994) constructed a model to capture the most important activities in the enterprise. Based on simulation execution, he presented some insight far more visible than mental models.

All these aforementioned architectures have produced many fine results, however, most of them are focused on formulating the data and information aspects of an enterprise. The knowledge is mixed with the information and treated as part of the information in processing and utility. The modelling language used by the above schemes is also suitable for representing data and information but not for expressing the knowledge (De Souza & Zhao, 1997*a*). These modelling schemes focus on precise and complete knowledge, or complete unknown random factors in the model. Little attention is paid to the imprecise and incomplete knowledge that persists in the enterprise (Asai, 1995; De Souza & Zhao, 1997*b*). Furthermore, these modelling techniques need to be improved in the consideration of simulation as it is an effective way to utilize the model.

Actually, the enterprise paradigm has shifted with advances in technology, from an original labor and physical facility orientation, through data and information, and currently onto the present knowledge intensive format. The knowledge embodied in an enterprise (common sense, experience, rules etc.) plays a significant role and often becomes the most important resource for the success of the modern enterprise (Gosset & Massotte, 1994; Zhao & De Souza, 1997). The functionality and the behavior of the enterprise are increasingly governed by the knowledge of these so-called "intelligent" enterprises. Therefore, to model the dominant and representative features of these enterprises, it can be argued that more emphasis ought to be placed on the knowledge instead of on the detailed data and information representations during modelling. Meanwhile, modelling at the knowledge level can yield many benefits: First, the enterprise of today are full of dynamic and uncertain components; the governing law of these components is heavily dependent on the vast body of knowledge, but not necessarily on the information and data. Modelling at the knowledge level can encompass the dynamic and uncertain aspects relatively easily. Second, modelling at this level can enable simple organization of information and data because the knowledge acts directions for further information and data. Representing the knowledge in the model is equivalent to capturing the kernel component of the information and data. Third, modelling at the knowledge level can utilize many well-developed powerful knowledge representational schemes to describe features and characteristics of an enterprise that cannot be easily represented, by conventional data and information representational schemes. Finally, the model should remain live and be validated interactively. Modelling at the knowledge level thus paves the way to conduct intelligent modelling by a variety of available learning techniques.

Within this context, the enterprise functionality and the business process will be precisely specified, analyzed, and modeled at the knowledge level. A systematic formalism to define activities and business processes is presented, as well as clear mathematical formulation and template representations. Moreover, an integration of multiple types of knowledge is explored. In conclusion, an application that summarizes the most obvious results of the research are presented.

2. KNOWLEDGE CLASSIFICATION IN AN ENTERPRISE

Modelling an enterprise requires sufficient understanding to represent its behavior and functionality effectively. If the enterprise behavior and functionality are completely known, namely, the knowledge of the enterprise is precise knowledge, then conventional precise knowledge representational methods such as deterministic mathematical equations, rules, facts, frames, logic, and the like can be used to explicitly form activities and processes, no matter whether they are in linguistic form or defined numerically. However, if enterprise functionality and behavior are completely unknown, namely, the knowledge of the enterprise is random knowledge, one feasible modelling approach for activities and processes is random representation on the basis of probability theory or stochastic theory. In reality, the understanding of enterprise functionality and behavior usually falls between these two polar states. The enterprise behavior and functionality are either partially known or undergo change. In other words, the knowledge of the enterprise is muddy knowledge, that is, neither precise nor random. Activities and processes cannot be precisely formulated by conventional precise knowledge representational methods, and conversely, a random representational approach will omit the known aspects and introduce extra uncertainty in the model. Therefore, vague, imprecise, ambiguous knowledge representational methods such as fuzzy logic, Bayesian probability, certainty factor models, etc. can be adopted. Conventional modelling methods in the CIM scenario reveal little, if anything, about this imprecise, vague ambiguous knowledge. In effect, the understanding and representational language of enterprise behavior and functionality constitute a continuous spectrum.

For a particular enterprise, the functionality and behavior usually contain the combination of much precise, muddy, and random knowledge. Further, this knowledge may manifest itself in two broad forms, either as numerical knowledge or symbolic knowledge. The numerical knowledge may appear in mathematical equations, regression models, stochastic models, as a Markov process, or as a data set. A trained neural network can also be used to represent input–

https://doi.org/10.1017/S0890060498121108 Published online by Cambridge University Press

output numerical relationships whose transformation process cannot be easily described by obvious mathematical formulae. On the contrary, the symbolic knowledge can be presented in linguistic, symbolic form such as facts, frames, semantics, production rules, logic, natural language, and so on. The above-mentioned precise, muddy, or random knowledge that appear in either symbolic or numerical form can crystallize into six categories of knowledge: precise numerical knowledge (such as mathematical function), precise symbolic knowledge (such as rules, facts), muddy numerical knowledge (such as Bayesian probability, certain factor theory), muddy symbolic knowledge (such as fuzzy knowledge), random numerical knowledge (such as stochastic or chaotic process), or random symbolic knowledge (such as random symbolic generation).

Normally, enterprise behavior and functionality contain varying degrees of these six types of knowledge and hence, the modelling needs an integration of these six categories of knowledge as shown in Eq. (1).

$$\text{Model} := \{\oplus | K_{PS}; K_{PN}; K_{MS}; K_{MS}; K_{RS}; K_{RN}\} \quad (1)$$

where K_{PS} denotes precise symbolic Knowledge; K_{PN}, precise numerical Knowledge; K_{MS}, muddy symbolic knowledge; K_{MN}, muddy numerical knowledge; K_{RS}, random symbolic knowledge; K_{RN}, random numerical knowledge; and "$\oplus$" represents the integration of these six categories of knowledge.

The integration "$\oplus$" should not be simply the lump sum to make an accumulated whole, but be carried out in a hybridization and coupling fashion to form a synergistic whole. One promising way to accomplish this action is to integrate the multiple types of knowledge at the knowledge level. In other words, using knowledge about the domain knowledge to accomplish "deep" level integration. The knowledge about the domain knowledge is meta knowledge (MK), namely, the knowledge concerning the propriety, characteristic, and utility of the domain knowledge. The integrated structure is shown in Figure 1. The function of meta knowledge is to use, control, manage, and coordinate the domain knowledge.

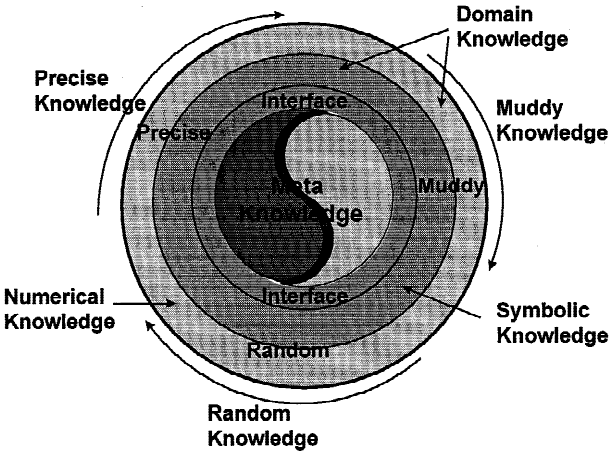

Fig. 1. The integration using meta system as a kernel.

3. ACTIVITY FORMALISM

3.1. Definition and graphic representation

In essence, an activity is defined as a transformation from input to output subject to some condition. It can also be denoted as a function that related the input state into an output state under some conditions. It can be written as Eq. (2),

$$\text{Output state} = f(\text{Input state}) \text{ subject to guard}(f) = C \quad (2)$$

Several formalisms for graphical representations of an activity have been proposed as shown in Figure 2. A generic activity model (GAM) has been proposed by ISO TC 184 (ISO, 1990). It makes use of a box with 10 legs as illustrated in Figure 2(a). It differentiates between the information flow and the material flow and makes for explicit resource input and output. However, the input and output have no precise semantics, and the model fails, in the authors' opinion, to describe the triggering condition as well as the ending status of activities. Furthermore, the representation is deemed too complex. Another simpler representation is the IDEF0 ICOM box depicted by Figure 2(b) based on actigrams of SADT (Vallespir et al., 1991). It is often used in practice to represent either processes or activities. However, it may be considered too simple. Another candidate for activity representation is the CIMOSA enterprise activity box. It has a graphical representation compliant to the ISO GAM representation and has semantically well-defined inputs and outputs [Fig. 2(c)] (Vernadat, 1995, 1996). However, it does not clearly represent the triggering condition and control input. Further, the knowledge characteristics and function in the activity is not explicitly specified. Hence, the authors have proposed a revised activity box as shown in Figure 2(d), which is a modification based on the CIMOSA representation. The difference between the proposed activity and that of CIMOSA lies in the fact that the control output in Figure 2(c) is modified to express the triggering condition of the activity as well as facilitate the recording of the termination status and subsequent postprocessing. The function input and output also contains knowledge views beyond the physical and information views.

In Figure 2(d), the function input (FI) refers to a set of object views to be transformed by the activity; while the function output (FO) is the one transformed by the activity. Generally, the object views offered by CIMOSA are referred to as the physical and the information views. The knowledge is mixed with the information and data in this information view (Vernadat, 1996). To emphasize the function of the knowledge, a knowledge view is added to separate the knowledge from the information in the object view. The function input and output are modified as the Physical View (PV), Information View (IV) and Knowledge (or de-

https://doi.org/10.1017/S0890060498121108 Published online by Cambridge University Press

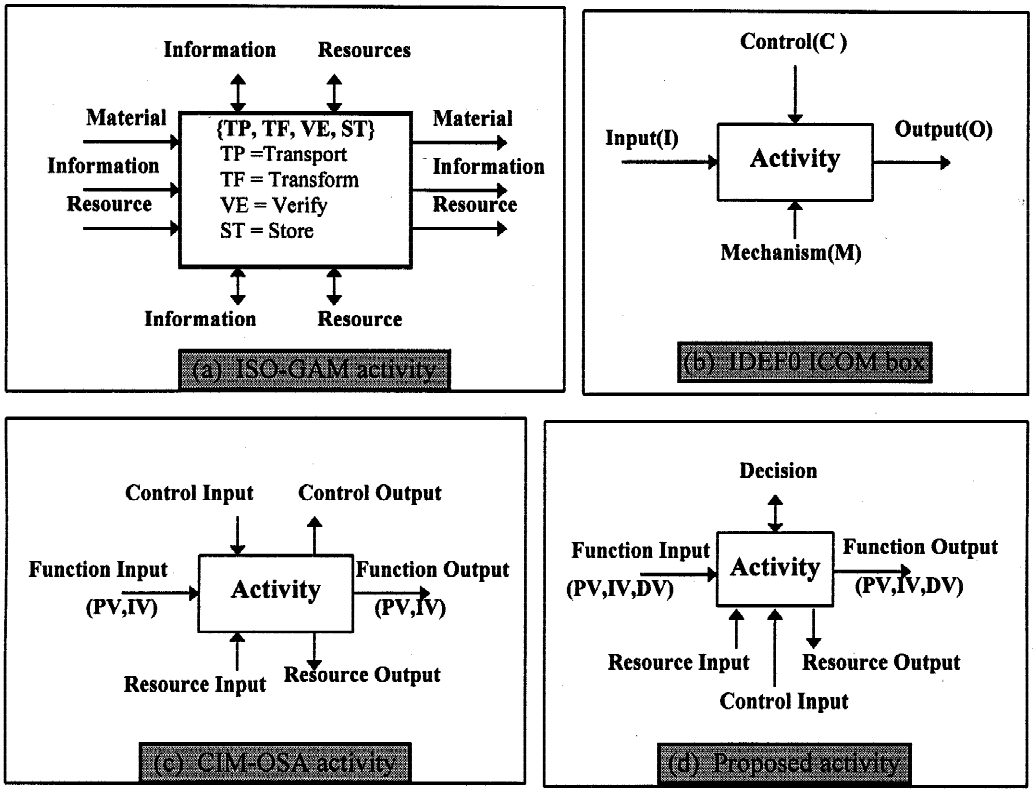

Fig. 2. The graphic representation of activity.

cision) View (KV). The Resource Input (RI) refers to a set of object views used as the resource to support the execution of the activity; it will not affect the transformation of the FI, FO; while the resource output (RO) is a set of object views used as resources to be transformed after execution of the activity. The resource object view encompasses the physical and information views. Control input (CI) refers to a set of information views used to support or constrain the execution but is not modified by the activity. The decision (D) contains the triggering condition for execution of the activity and is postprocessed after the activity runs. It will not be affected by execution of the activity. It is mainly concerned with conditions, constraints to perform the activity, and termination status of the execution.

3.2. Mathematical description

Although graphical notations are useful to be understood; the mathematical expression can give more precise and deep insight about the nature of the activities. Moreover, it can offer many tools to model the process with the aid of many fine mathematical results.

Essentially, an activity can be expressed as follows:

$$\boldsymbol{Q_A} = \delta(D.\text{triggering\_port})\, \boldsymbol{f}(\boldsymbol{I_A}, \boldsymbol{fe}, t) \tag{3}$$

where $\boldsymbol{Q_A} \in \boldsymbol{R}^{2\times 3}$ is the output vector of the activity $\boldsymbol{Q_A} = (\boldsymbol{F_O}, \boldsymbol{R_O},)^{\mathrm{T}}$. $\boldsymbol{I_A} \in \boldsymbol{R}^{3\times 3}$ is an input vector of the activity $\boldsymbol{I_A} = (\boldsymbol{F_I}, \boldsymbol{R_I}, \boldsymbol{C_I})^{\mathrm{T}}$; t $\in \boldsymbol{R}$ is time; $\boldsymbol{f} \in \boldsymbol{R}^{3\times 3} \times \boldsymbol{R}^n \times \boldsymbol{R} \to \boldsymbol{R}^{2\times 3}$ is a mapping field of the activity; $\delta(\cdot)$ is delta function in term of triggering condition in decision,

$$\delta_{\mathrm{D}.Trigger\_port} = \begin{cases} 1 \text{ if } D.\ Triggering\_port \text{ is } True \\ 0 \text{ if } D.\ Triggering\_port \text{ is } False \end{cases} \tag{10}$$

D is decision and

$$\mathrm{D} = \{\text{Triggering\_port; Post\_processor}\}; \tag{4}$$

$\boldsymbol{fe}$ is a user-defined feature set to represent features in the activity. It is intermediate variable $\boldsymbol{fe} \in \boldsymbol{R}^n = (\mathit{feature}_1, \ldots, \mathit{feature}_n)$,

$$\boldsymbol{fe} = \boldsymbol{g}(\boldsymbol{I_A}, t) \tag{5}$$

in which $\boldsymbol{g} \in \boldsymbol{R}^3 \times \boldsymbol{R} \to \boldsymbol{R}^{\mathrm{n}}$ is a mapping from function input to the defined feature set.

$$\boldsymbol{F} = (F_{\mathrm{PV}}, F_{\mathrm{IV}}, F_{\mathrm{KV}})^{\mathrm{T}}; \tag{6}$$

$$\boldsymbol{R} = (R_{\mathrm{PV}}, R_{\mathrm{IV}}, R_{\mathrm{KV}})^{\mathrm{T}}; \tag{7}$$

$$\boldsymbol{C} = (C_{\mathrm{IV}}, C_{\mathrm{KV}})^{\mathrm{T}}; \tag{8}$$

https://doi.org/10.1017/S0890060498121108 Published online by Cambridge University Press

where F_{PV}, F_{IV}, and F_{KV} denote the functional physical view, information view and the knowledge view, respectively. R_{PV}, R_{IV} and R_{KV} is resource physical, information and knowledge view, respectively. The mapping cannot be concisely expressed as a numerical equation further, instead, a composite form containing multiple types of knowledge should be used. In other words, the mapping should be conducted by the integration of a large amount of knowledge with integrated structure as shown in Eq. (1), namely,

$$\boldsymbol{f} := \{K_{PS} \oplus K_{PN} \oplus K_{MS} \oplus K_{RS} \oplus K_{RN}\} \tag{9}$$

If $\boldsymbol{f}(\cdot)$ can well be constructed in either symbolic or numerical way, then the activity is called *structured activity*, otherwise, the activity is *nonconstructed activity*. If the mapping function $\boldsymbol{f}(\cdot)$ can be constructed as a set of mathematical equation, then the activity is defined as a *numerical activity*, otherwise, it is named *composite activity*. Usually, the constructed activity is solvable and the numerical one can obtain an optimal solution. The nonconstructed activity is usually hard to be solved and in solution multiple tools, including mathematical computation and symbolic reasoning need to be used.

In the case that the mapping field $\boldsymbol{f}(\cdot)$ is an unit mapping field, namely, $\boldsymbol{f}(\cdot) = \mathbf{I}$, the resource will never be changed and the function input equals the function output. This means the activity is just a delay. If an Eq. (2) can be transformed as

$$\boldsymbol{I_A} = \delta(D_{\cdot\text{trigger\_condition}})\boldsymbol{f}^{-1}(\boldsymbol{O_A}, t). \tag{11}$$

This implies that the activity is inevitable. As an example, consider an activity that conducts order evaluation. The activity function input is the order and the output is the required production capability. Hence, the activity taking production capability as input and the kind of order that can be satisfied as the output is thus the inverse of the former.

3.3. Activity specification

Enterprise activities are used to describe functionality with an allocation of resource and decision support. In CIMOSA, the template representation uses Pascal-like activity behavior. This representation is complex and systematically incomplete (Nager et al., 1995). Hence, the template representation of activities is enhanced. Its specification is characterized by three components as shown in Eq. (12).

$$\text{Activity} := \{\text{Header; Decision; Body}\}. \tag{12}$$

The Header comprises the description of the activity including activity name, activity type, design authority, and explanation to the activity. The Decision is the lever to control and regulate the activity, it contains a Triggering_port to invoke the activity and a Post_processor that postprocesses after execution of the activity. The Triggering_port is a set of filters expressed by triggering rules to decide and restrict the execution of the activity. The Post_processor is a set of operations that need to be conducted after the activity. A typical triggering rule and postprocessor is given in Figure 3.

The body declares the behavior of the activity, it is comprised of the input, output, and mapping field as in Eq. (13).

$$\text{Body} := \{\text{Input; Output; Mapping\_field}\}, \tag{13}$$

where input contains FI, RI, and CI of an activity. Output contains FO, RO, and features (Fe) of the activity. The Mapping_field specifies various functions which transform the Input (FI, RI, CI) to the Output (FO, RO, CO).

The mapping field is an integration of six types of knowledge as previously mentioned in Eq. (8). The precise symbolic mapping knowledge uses frames, facts, rules, procedures, and operations as representation mechanisms. The representational template and its explanation is in Figure 4.

The precise numerical mapping knowledge contains a mathematical function or a subnumerical computing routine. It can generically be expressed as

$$\langle output1, \ldots, output\ n\rangle = \langle FUN\rangle\ \langle input1, \ldots, input\ m\rangle\ or$$

$$output := CALL\ \langle name\rangle\ \langle input1, \ldots, input\ n\rangle\},$$

where *output* is a string; *input* is a string, real or integer; *FUN* may be a mathematical function $f(\square)$, it may also be a trained neural network $NN(\square)$, or auto regression model $AR(\square)$, $ARMA(\square)$ from the known data. *Name* is a string.

The muddy numerical mapping knowledge is the standard Baysian probability or certain factor model referred to by Krause and Clark (1993). The muddy symbolic mapping knowledge consists of fuzzy facts, fuzzy rules, and fuzzy procedures. They are often in the form shown in Figure 5.

```
                    ⋮
TR i: WHEN <condition 1> and/or … and/or <condition n>
      START ACTIVITY  /* ith triggering rule to invoke the activity when
                         the condition 1 to condition n are satisfied */
                    ⋮
PPS j: <operator>(name)     /* jth post processor
where the operator := {DELETE , /* delete a results */
                       SAVE,    /* store a results */
                       SET,     /* set or change a results */
                       MODIFY, /* modify a results */
                       … }
                    ⋮
```

Fig. 3. The representational language of decision of an activity.

https://doi.org/10.1017/S0890060498121108 Published online by Cambridge University Press

```
                                        ⋮
Frame i: string /* ith frame */
Superclass: string /* the father frame */
Subclass: string /* the child frame */
  {
     ⋮
  slot i: string /* one slot of the frame */
  facet: string /* type of the slot */
  value: <real, integer, string, rule set, operation, method> /* the value of the slot */
     ⋮
  }
                                        ⋮
RuleSet j: { /* one rule set */
                ⋮
/* production rules: map if-part to then-part */
    IF <expression 1> <connective> <value 1> and/or;…;
       <expression n> <connective> <value n>
    THEN <expression 1> <connective> <value 1> <Dos> and/or;…;
       <expression m> <connective><value m> <Dos>.
                ⋮
/* sequential rules: a series of actions will be executed when condition is true */
    WHEN <condition> SEQDO {<action 1>, …, <action n>};
                ⋮
/* selective rules: perform two different actions */
    IF <condition> DO {<action>}
    ELSE DO {<action> };
                ⋮
/* repeatative rules: perform actions repetitively until condition is satisfied*/
    WHILE <condition> DO { <action>};
                ⋮
  } /* end of the rule set i */
Operation j: <operator><name>
where operator := {SAVE,     /*save a results */
                   DO,       /*run a procedure */
                   DELETE, /* delete a result */
                   CREATE, /*create a results */
                   CALL      /* call a sub-procedure */}
                                        ⋮
/* where condition i:= {string}, connective := {is,do,=,!=,>,<,>=,<=}, value := {string,
integer, real}; &:=and; |:=or; Dos := {real, integer} is degree of support. action is a
operation such as SAVE, DO, DELETE, CREATE,....*/
```

Fig. 4. The representation of precise symbolic mapping.

https://doi.org/10.1017/S0890060498121108 Published online by Cambridge University Press

```
                                                    ⋮
Fframe i: string   /* fuzzy frame i */
Superfclass: string   /* the father frame */
Subfclass: string   /* the child frame */
        {
                ⋮
        fslot j: string  /* the jth slot of this frame */
        ffacet:  connective  /* the type of the slot */
        fvalue: <fuzzy facts, fuzzy rule set, fuzzy evaluation> /* the value of the slot */
                  ⋮
        }
                                                    ⋮
        /* fuzzy evaluation: evaluate the fuzzy value from n inputs fuzzy variables */
FE j: <fuzzy_variable> = <FFUN>(Fuzzy_input1, ..., Fuzzy_inputn);
        /* where FFUN is fuzzy operations such as fuzzy sum (FSUM), fuzzy minus (FMINUS),
        fuzzy multiplity (FMULTPILITY), integral (FINTEGRAL), etc.*/
                                                    ⋮
        /* fuzzy rules: map  if-part to then part */
FRULE k:
   Fuzzification := {triangular | bell-shape | trop } /* the fuzzification method */
   DeFuzzification := {centroid | height | maxmin} /* defuzzification method */
   IF <fuzzy_variable 1> <connective> <fuzzy_value 1> and(or) ;…;
      <fuzzy_variable n> <connective> <fuzzy_value n>
 THEN <fuzzy_variable 1> <connective> <fuzzy_value 1>  and(or) ;…;
           <fuzzy_variable n> <connective><fuzzy_value n>.
                                                    ⋮
 IF <operator> <parameters> <fuzzy_value> and(or)
        <operator> <parameters> <fuzzy_value>
THEN <operator><parameters> <fuzzy_value> and(or);…
     ELSE <operator><parameters><fuzzy_value>}
                                                    ⋮
          /* fuzzy procedure represents a series of executions */
FP l:
  <Action> =  <Action1> (Fuziness) Before <Action j> (Fuziness)
/* where fuzzy_variable i: is string, connective is, " is, do, =,!=,>, <, >=, <=", fuzzy_value
i:= {string} is the fuzzy level description. & := and; | := or;. Dos := {real, integer} is  degree
of support.*/
```

Fig. 5. The representation of muddy symbolic mapping.

https://doi.org/10.1017/S0890060498121108 Published online by Cambridge University Press

The random mapping knowledge contains random manipulation and random variable generation. The random manipulation is used to compute the randomness of variables where the random symbolic mapping knowledge produces random symbolic variables and the random numerical knowledge produces numerical variables. The template for random knowledge is represented as in Figure 6.

The meta knowledge contains the functional description, constraint, and control rules. Functional descriptions are used to describe the function or the work that can be performed for each type of domain knowledge. Constraints are used to describe the requirement of accomplishing the above function for each system. Control rules are used to control the utility of a wide range of knowledge to obtain the output from the input. It contains rules and procedures. The meta knowledge is as shown in Figure 7.

4. THE BUSINESS PROCESS

The business process is used to describe the enterprise behavior. It is aggregated by a series of activities and only triggered by events such as order arrival, machine breakdown etc. In theory, it has been illustrated that activities can be combined into processes by using a parallel operator denoted as "$\|$" to describe any flow of control assuming that this operator is not commutative (Curtis et al., 1992). In CIMOSA, a process is aggregated by a control structure rule set (procedural rule set) such as sequence, parallelism, rendezvous, choice, or loop to connect activities in a network fashion (Vernadat, 1995). However, many business processes in manufacturing enterprises go hand-in-hand with a vast body of incomplete and vague knowledge. The connection and interaction of activities is reached in an imprecise fuzzy relationship instead of as definitive complete knowledge. Furthermore, some connections of the activities may be on the basis of stochastic relations. In short, it is strongly dependent on multiple types of knowledge. Therefore, its representational formalism should adopt a similar structure as shown in Figure 1 from the knowledge perspective. In other words, although the meta system and domain knowledge are different in the business process and enterprise activity, the integrated structure is the same.

In a general mathematical form, the process can be expressed as

$$\mathrm{P} = \delta(\mathrm{D.condition})\boldsymbol{f}(\boldsymbol{EA}, t), \tag{14}$$

where D denotes the decision that is shown in Eq. (4). The Triggering condition is event driven, and the End_processor performs the post operations after the business process such as storing results and recording termination status. $EA \in R^n$ is the enterprise activity vector and $EA = (\mathrm{EA}_1, \ldots, \mathrm{EA}_n)^{\mathrm{T}}$. EAi represents the ith enterprise activity. $\boldsymbol{f}(\cdot) \in \boldsymbol{R}^n \times \boldsymbol{R} \rightarrow \boldsymbol{R}$ is the mapping field which is constructed by an integration of the multiple categories of mappings as shown in Eq. (1). The $\delta(\cdot)$ is defined by Eq. (10).

In a more detailed form, a business process also consists of three components as shown in Eq. (13). The header defines the name, type, and illustration of the process. The decision contains the triggering condition for execution of the business process and post processes sequences. Its structure is the same with Eq. (4) except that it is for a process. The body specifies the behavior of the business process and is constructed of multiple types of mappings. The process is at a high level and operates on the activities. Correspondingly, the knowledge is also different. For precise symbolic

```
⋮
/* randomness manipulation function */
RF i: <output> = <PFUN><input1, ..., input n>
/* where output := {string}, input := {string, real, integer}, PFUN is
random function. */
⋮
/* random variables generator */
<Random_variable> IS <name>< random value>;
⋮
```

Fig. 6. The representation of random mapping.

https://doi.org/10.1017/S0890060498121108 Published online by Cambridge University Press

⋮

/ The function of each type of knowledge */*

K_{PN} : <variable1, variable 2, …, variable n >

⋮

/ constraint states that proposition j needs the propositions 1 to propostition n*/*

CST i: <proposition> REQUIRE *<proposition 1>* |; …; *<proposition n>*

⋮

/* conflict resolver : resolve the conflict of n variables */

CR j: *<variable>* = <RESOLVER><*variable1*,…,*variable n*>

/ where RESOLVER is COMBINE, REPLACEMENT, PRIORITY, DELETE, AND, or OR, etc.. */*

⋮

*/*conditional rules: perform action if a condition is satisfied */*

Rule i: IF *<premise>* DO *<action>*

⋮

/ requirement rules: the premier needs the operation perform until the condition is satisfied */*

Rule j: {*<premise>*} NEEDS{ DO { <operation>} WHILE {<condition>}}

⋮

/ finish the process when the condition is satisfied */*

WHEN *<condition>* FINISH

/ where premise := {string}; proposition := {string}; &: = and; | := or.}*/*

Fig. 7. The representation of meta knowledge.

knowledge, it usually contains various rules such as production, sequential, conditional or spawning rule, loop, termination rule or constraint. The representation of the above mentioned rules can be expressed as in Figure 8.

The fuzzy knowledge contains fuzzy connections of activities as shown in Figure 9. Other types of knowledge are similar with that in the activity.

5. IMPLEMENTATION OF A MODELLING PLATFORM

Based on the aforementioned knowledge intensive modelling technique, a modelling platform prototype is developed. The structural diagram of the modelling platform can be illustrated as in Figure 10.

The modelling platform consists of an operational-level modelling tool and an organizational level modelling tool. The operational level modelling tool is realized by using Extend™ (1995) simulation software, which provides a C-type language to develop more sophisticated knowledge-based functions and activities. The organizational level modelling tool is developed with visual basic. The modelling library consists of neural network, fuzzy logic, genetic algorithm, mathematical analysis, and other available software. Some components in the modelling library, such as the neural network, the fuzzy logic and genetic algorithm, are written with visual basic, and some available software are directly integrated into the modelling platform through dynamic-link library (DDL), open data base connectivity (ODBC), dynamic data exchange (DDE), or object linking and embedding

https://doi.org/10.1017/S0890060498121108 Published online by Cambridge University Press

Production rule: IF <*condition 1*>&(|), …, <*condition n*>

THEN <*result1*> &(|), …, <*result n*>

Conditional rule: WHEN (*condition*) DO EA_i

Sequential rule: WHEN (*condition*) SEQDO {EA_1; …;EA_n;}

Spawning rule: WHEN (*condition*) DO { EA_1 &…& EA_n }

WHEN (*condition*) DO {EA_1 |… | EA_n}

Loop: DO { EA_i } WHILE (*condition*)

Termination rule: WHEN (*condition*) DO FINISH

CONSTRAINT := { EA_i <*connective*> EA_j}

/ where connective := { BEFORE; AFTER; DURING;}. & represents that the activity needs to start simultaneously and | is the operation selected from the alternatives. */*

Fig. 8. The representation of precise symbolic mapping of a business process.

(OLE). The meta system shell is written in visual basic and used to manage and control the organizational, operational tools and the modelling library.

The modelling platform relies on a graphic icon library of predefined enterprise objects to represent business processes and enterprise activities. The user draws the process icon or activity icon from the library and inputs the related knowledge into the process or activity. The user interface of the modelling platform is shown in Figure 11. After the model is built, it can be used for simulation to analyze and optimize the designed enterprise.

6. EXAMPLE

Let us consider an activity that represents machine maintenance depicted by Figure 12. The machine maintenance activity is performed by a team of workers and consists of three basic steps. The FI consists of the faulty machine and the associated information and knowledge about the faulty machine such as the machine fault times, composition of the machine etc. The FO contains the repaired machine and the corresponding knowledge and information about the machine. This activity needs a support resource such as la-

⋮

Fuzzification := {triangular | bell-shape | trop } */* fuzzification method */*

DeFuzzifMethod := {centroid | height | maxmin} */*Defuzzification method*/*

WHEN(<*fuzzy_variable*> <*connective*> <*fuzzy_value*>) DO EA_i;

<EA_i><*connective*><EA_j> <fuzzy_calue>}

⋮

/ where connective := {BEFORE, AFTER, DURING} */*

Fig. 9. The representation of muddy symbolic mapping of a business process.

https://doi.org/10.1017/S0890060498121108 Published online by Cambridge University Press

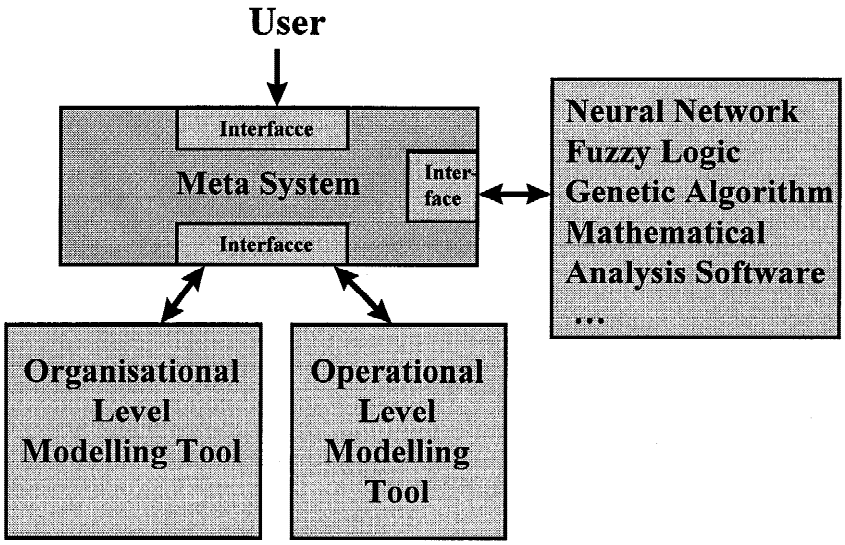

Fig. 10. Structural diagram of the modelling platform.

borer, fault analysis equipment, software, and maintenance tools. The CI is necessary for activity execution but will not be changed by the activity. For example, the machine designed life, the function of the machine in the production line, machine worked time are the CIs. The Decision includes the triggering rules, such as when the machine is detected at fault and needs to be repaired, when the activity starts, and the postprocessor is activated for such tasks as recording the termination status, storing of the results, and so on. The mapping field consists of fuzzy knowledge such as maintenance time, which has a fuzzy relationship with the worker skill, repair results, and testing; the stochastic knowledge includes the processing time that exists randomly to some degree; the deterministic numerical knowl-

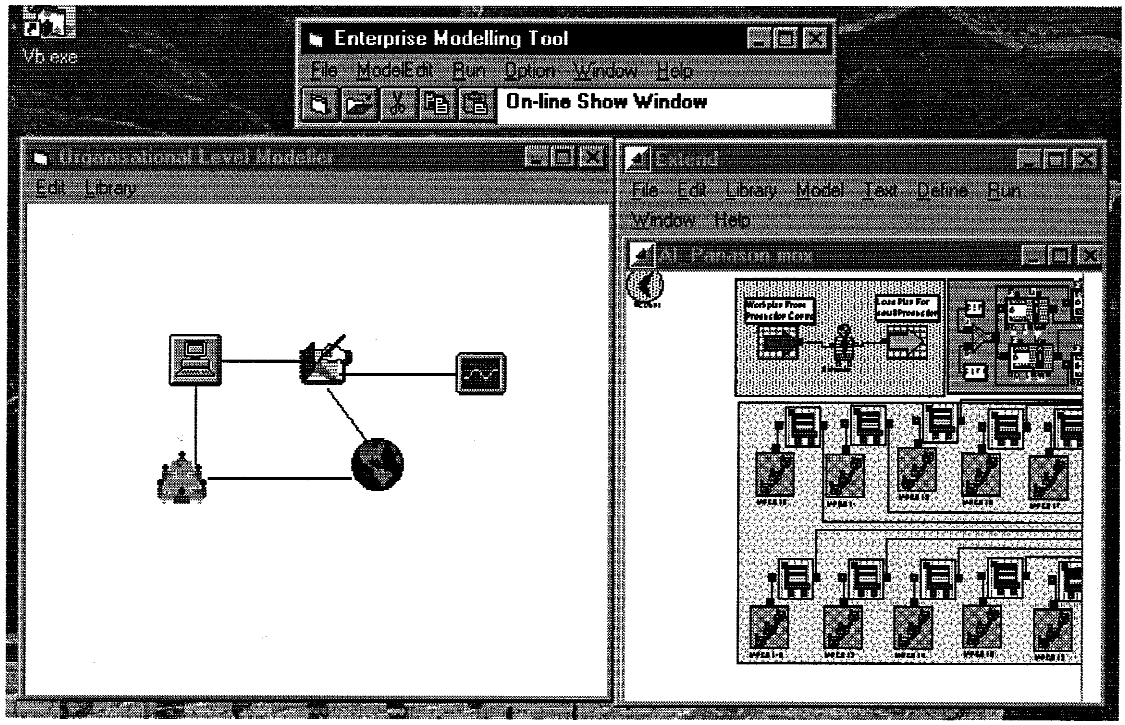

Fig. 11. The user interface of the modelling platform.

https://doi.org/10.1017/S0890060498121108 Published online by Cambridge University Press

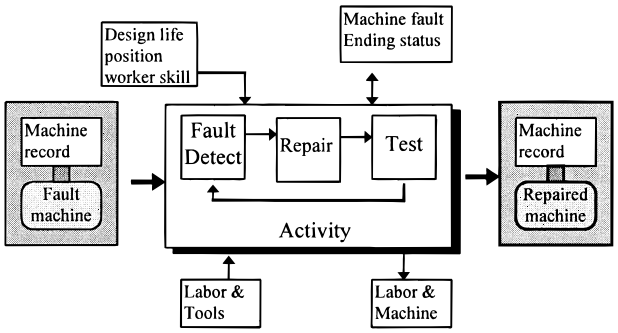

Fig. 12. Machine maintenance process.

edge includes the signal processing and fault tree reasoning; the symbolic knowledge includes fault position reasoning based on the fault tree, and the maintenance results analyzed by reasoning. The machine maintenance function is represented as an icon as shown in Figure 13.

The following is a part description of the maintenance activity in the icon:

ENTERPRISE ACTIVITY i

/* *Header: the name, type, identifier and explanation of this activity* */

Name: machine-maintenance

Type: man-machine

Identifier: EAi

Description: This activity is used to represent a maintenance process

/* *Decision: the activity to decide the execution of this activity* */

Decision: {

IF machine is fault and worth repairing THEN START ACTIVITY;

ELSE RETURN Activity cannot be executed; /* *execution condition of the activity* */

⋮

/* Decision: *post processing after activity* */

WHEN repaired_results = FINISH DO {

Termination_status = repaired_status;

Fig. 13. The modelling icon of machine maintenance activity.

SAVE(results);

…;}

}

/* *Body: inputs and outputs of this activity* */

Function Input: FI.PV, FI.IV, FI.KV

/* *PV, IV and KV of function input such as fault machine, machine fault times, etc.* */

Function Output: FO.PV, FO.IV, FO.KV

/* *PV, IV and KV of function output such as repaired machine, machine fault times, etc.* */

Resource Input: RI.PV, RI.IV

/* *PV, IV of resource input such as labour, tools, software, recording, etc.* */

Resource Output: RO.PV, RO.IV

/* *PV, IV of resource output such as labour, tools, software, recording, etc.* */

Control Input: CI.IV

/* *IV of control input such as machine design life, machine importance, etc.* */

Features: actual_processing_time, fault_rate;

/* *the main features of this activity* */

/* Body: *the integration of knowledge to map the inputs to outputs* */

Activity Knowledge:{

Precise Symbolic Knowledge {/* *knowledge which can be precisely defined* */

RO.PV.workers = RI.PV.workers; /* workers cannot be changed */

⋮

FO.PV SEQDO {fault_identify; repair; test};

/* *to repair the machine needs to perform fault identification, repair and test operation* */

Fault_identify NEED SEQDO {sampling,

signal_processing,

signal_identification,

fault_location_detect.}

/* *fault identification needs signal processing, data analysis and fault detect* */

sampling REQUIRE {RI.PV.AD_card, RI.PV.Computer}

⋮

IF test.status = OK THEN {/* *check whether the repair is successful or not* */

FO.PV.machine = repaired machine; /* *if successful, machine become ok* */

RO.IV.used_time = RI.IV.used_time + processing_time;

https://doi.org/10.1017/S0890060498121108 Published online by Cambridge University Press

```
        /* record the time information of the maintenance
        equipment */
        repaired_status = OK;}
    ELSE repaired_status = FALSE;
      /* unsuccessful repair */
    … }
Precise Numerical Knowledge {
    FO.KV.fault_times = FI.KV.fault_times +1; /* the fault
    time of machine plus 1 */
    FI.IV.Max_processing_time = 50.0; /* the max value for
    maintenance process time */
    FI.IV.Min_processing_time = 19.0; /* the min value for
    maintenance process time */
    Fe.actual_processing_time = FI.KV.processing_time +
    pt_random;
      /* actual maintenance time should be the process time
      plus a random value */
    signal = sampling; /* sampling to obtain the signal */
    FO.IV.spectrum = FFT(signal); /* perform Fast Fourier
    Transform */
        ⋮
        }
Muddy Symbolic Knowledge: {
    FO.KV.Fuzzification = FI.KV.triangular; /* select the
    fuzzification method */
    FO.KV.Defuzzification_method = FI.KV.Centroid; /* se-
    lect the de-fuzzification method */
        ⋮
      /* fuzzy rules: the maintenance process time with re-
      spect to the machine complexity */
      IF FI.PV.machine is very_complex THEN FO.KV.pro-
      cessing_time is very_long;
    IF FI.PV.machine is complex THEN FO.KV.process-
    ing_time is long;
    IF FI.PV.machine is normal THEN FO.KV.process-
    ing_time is middle;
    IF FI.PV.machine is simple THEN FO.KV.process-
    ing_time is short;
    IF FI.PV.machine is very_simple THEN FO.KV.process-
    ing_time is very_short;
        ⋮
    … }
Random Knowledge {
    /* manipulate and generate random variable "pt_ran-
    dom" */
        ⋮
    normal_distribution = normal(FI.IV.mean, FI.IV.vari-
    ant);
    FO.IV.pt_random = GENERATE(normal_distribution);
        ⋮
… }
Meta Knowledge { /* The function of each types of knowl-
edge */
    PK: {fault_identify, maintenance} /* precise knowledge
    */
    MK: {processing_time, fault_times, actual_process-
    ing_time, …} /* muddy knowledge */
    RK: {pt_random} /* random knowledge */
    CONSTRAINT: {/* some constraint for operation order
    */
      Decision.repaired_status NEED {fault_detect & main-
      tenance & test}
      fault_detect AFTER signal_processing;
      fault_detect USE (diagnosis_method 1 | diagno-
      sis_method2);
      diagnosis_method1 NEED {single_processing &
      power_spectrum &diagnosis}
        ⋮
      }
Conflict_Resolver: {diagnosis_results =
    OR(diagnosis_results_from_method1, diagnosis_re-
    sults_from_method2};
    /* In the case that conflict exist between diagnose results
    form different method, the worst situation is considered
    */
WHEN activity = START {/* when the activity start, the
features are evaluated */
    Fe = MAPPING(RI, FI, CI);
    (FO,RO) = MAPPING(FI,RI,CI,Fe);
      /* the function output and resource output are evalu-
      ated */
        ⋮
      }
  }
END ACTIVITY
```

7. CONCLUSION

Business process and enterprise activity serve as a basis for modelling enterprise behavior and functionality in the context of CIM. As knowledge becomes the most important asset of today's enterprises, modelling at the knowledge level can be simplified. The modelling process, if made systematic and capable enough, can capture the dominant features of enterprises within the model. In this paper, a formalism to represent enterprise activities and business processes at the knowledge level has been specified based on the CIMOSA results. The representation involved the integra-

https://doi.org/10.1017/S0890060498121108 Published online by Cambridge University Press

tion of multiple types knowledge. The meta system as a kernel to accomplish the integration was illustrated as an efficient structure to achieve "deep" level integration.

ACKNOWLEDGMENTS

Each of the members of the Intelligent Manufacturing Research Group at the Centre for Engineering and Technology Management and discussions with the industrial collaborators are gratefully acknowledged, as is the support of the university in funding this activity.

REFERENCES

Asai, K. (1995). *Fuzzy Systems for Management*. IOS Press, Van Diemenstraat, Netherlands.

Bernus, P., & Nemes, L. (1995). Enterprise integration—Engineering tools for designing enterprise. *Proc. IFIP TC5 Working Conf. Models Methodol. Enterprise Integration*, 3–11. Queensland, Australia.

Curtis, B., Kellner, M.I., & Over, J. (1992). Process modelling. *Community ACM 35 (9)*, 75–90.

De Souza, R., & Zhao, Z.-Y. (1997*a)*. A case for knowledge intensive representation of dynamic resources in simulation. *Int. J. Production Res.* (accepted).

De Souza, R., & Zhao, Z.-Y. (1997*b)*. Intelligent representation of simulation resources. *Proc. ICPR '97*, 786–790. Osaka, Japan.

De Oliveria, A. (1994). AICIME—A technology and knowledge transfer methodology supported by industrial organization reengineering and an advanced CIME architecture based on ESPRIT products. *Proc. Tenth CIM-Europe Ann. Conf.*, 1024–1028. Copenhagen, Denmark.

Doumeingts, G., Vallespir, B., Darracar, D, & Roboam, M. (1987). Design methodology for advanced manufacturing systems. *Comput Industry 9(4)*, 271–296.

Extend™ (1995). *User's Manual for Extend v.3.2*. Image That, Inc., California.

Gossett, P., & Massotte, P. (1994). Knowledge systematisation and sharing—GNO—SIS(IMS) impact on industry. *Proc. Tenth CIM-Europe Ann. Conf.* Copenhagen, Denmark.

ISO (1990). Reference model for shop floor production standards. Technical Report No. 10314.

Krause, P., & Clark, D. (1993). *Representing Uncertain Knowledge*. Intellect Ltd. UK.

Moseley, L., & Dobson, O. (1996). Knowledge representation for reasoning: Tables, frames, and rules in a cutting fluids application. *AI EDAM 10(1)*, 37–46.

Marcos, W.C., Aguiar, R.H., & Weston, S.R. (1993). Model based approach supporting the life cycle of integrated manufacturing enterprises. *Proc. ICCIM'93*, 456–461. Singapore.

Mujataba, S.M. (1994). Enterprise modelling and simulation: Complex dynamic behaviour of a simple model of manufacturing. *Hewlett Packard J. 1*, 26–31.

Nager, G. (1995). Structuring and configuration of CIM systems for branch-specific medium-sized enterprise. *J Intell. Manufacturing 6*.

Rathwell, G.A., & Williams, T.J. (1995). Use of the Purdue Enterprise Reference Architecture and Methodology in industry. *Proc. IFIP TC5 Working Conf. Model and Methodologies for Enterprise Integration*, 12–44. Queensland, Australia.

Scheer, W.A., & Kruse, C. (1994). ARIS—Framework and toolset: A comprehensive business process re-engineering methodology. *Proc. Third Int. Conf. Automation, Robotics and Computer Vision (ICARCV'94)*. Singapore, 327–331.

Vernadat, F.B. (1995). CIM business process and enterprise activity modelling. *Proc. IFIP TC5 Working Conf. Model and Methodologies for Enterprise Integration*. Queensland, Australia.

Vernadat, F.B. (1996). *Enterprise Modelling and Integration*. Chapman & Hall, London.

Vallespir, B., Chen, D., Zanettin, M., & Doumeingts, G. (1991). Definition of a CIM architecture within the ESPRIT project IMPACS. In *Computer Applications in Production Engineering: Integration Aspects*. pp. 731–738. Elsevier, Amsterdam.

Williams, T.J. (1994). Architecture for integrating manufacturing activities and enterprises. *Comput. Industry 24(2–3)*, 111–140.

Zhao, Z.-Y., & De Souza, R. (1997). On improving the performance of hard disk drive final assembly via knowledge intensive simulation. *J. Electronics Manufacturing* (accepted).