Composing Enterprise Models:The Extended and The Virtual Enterprise

T. Janowski, G. Giménez Lugo[*] *and Zheng H.*[**]
The United Nations University
International Institute for Software Technology
P.O. Box 3058, Macau. E-mail: tj@iist.unu.edu

Abstract
This paper is a contribution to the semantics of the emerging discipline of enterprise engineering. We study the composition of models of individual enterprises into the model which represents the behaviour of an extended or a virtual enterprise. The former corresponds intuitively to the union of models: all activities taking place within and between individual enterprises. The latter to intersection: coordinated and shared activities which utilise the resources of all participating enterprises. Modelling adopts a unifying business perspective upon a firm (a discrete parts manufacturer), its structure (available resources) and behaviour (activities which utilise resources). Model composition is based on formal semantics. The result is a precise technical meaning for an extended and a virtual enterprise, suitable for symbolic execution, reasoning and foremost for understanding the difference between both concepts.

Keywords
Enterprise Modelling and Integration, Enterprise Engineering; Virtual Enterprise, Extended Enterprise, Formal Semantics, Abstraction and Composition

1 INTRODUCTION

The context for this work is the well-known instability of the manufacturing world. Enterprises must have enough flexibility to adapt quickly to changing environment conditions, enough know-how to manufacture advanced products,

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35390-6_58

enough resources to compete on the global market. They must revise the ways they run their business and cooperate with each other, by sharing or outsourcing costly or non-essential operations within an extended/virtual enterprise (Browne 1995) (Davidow and Malone 1992).

Since its introduction, the concept of an extended/virtual enterprise has received a lot of interest in the manufacturing community. We believe that the useful discussion about organisation, management (Pant and Hsu 1996) and implementation issues (Hardwick at al. 1997) should be supplemented by the more basic study about the meaning of this concept(s). We think such a study, cast in the framework of formal semantics, is needed for inclusion of an extended/virtual enterprise within the discipline of enterprise engineering, as well as contribute to the development of this discipline (Bernus and Nemes 1995, Vernadat 1996). This work is a step in this direction.

We shall take a unifying perspective upon a manufacturing firm described by its structure (available resources) and behaviour (activities which utilise resources). The resources are products (stocks), places to store them (warehouse) and ways to assemble them together (shopfloor) according to the bill of products. The activities include buying, selling or transforming products, with corresponding effects on the stocks, and expanding/reducing storing or production capacities. The activities are put together within a business process which is like a 'program' executing on the enterprise resources. Constructs for building processes include sequential composition, conditionals and repetitions (both employ properties over the state of resources), and alternatives which are decided by the environment (the market which allows to buy/sell products). The enterprise includes a set of processes executed concurrently, subject to external restrictions upon buying and selling and internal restrictions upon resources; competition for shared resources is resolved by priorities on processes. The model is defined formally. Its semantics is explained but not formalised. Exceeding by far the size of this article, formal semantics is subject of a companion paper (Janowski et.al. 1998).

We study two ways to put such models together. The first is the 'union' of models. This corresponds to an extended enterprise where we add together all resources and processes, allowing for complementary external activities (buying and selling) to take place as a synchronisation between enterprises. The effect is an internal relocation of products between participating enterprises. We also define an abstraction function from a collection of enterprises to a simple enterprise, to hide such a relocation. Along with the union we define the product of models. To this end we allow processes inside the enterprise to refer to the local as well as remote resources, owned by another enterprises (part of the extended enterprise). The product is the set of such processes which operate upon the resources of more than their own enterprise. We interpret this set as a virtual enterprise. This virtual enterprise has no resources of its own (it is "virtual") but is truly an abstraction of the activities which are shared and coordinated between enterprises.

The enterprise model is introduced and explained in Section 2. Section 3 is about composition of models, into the extended (Section 3.1) and virtual enterprise (Section 3.2). Section 4 contains summary and conclusions.

2 MODELLING AN ENTERPRISE

We present a simple model for a discrete parts manufacturer. Modelling adopts a unifying business perspective where many aspects of the firm are abstracted away (Figure 1). Part of the presentation is formal, part are informal explanations. We apply the notation of RAISE (Rigorous Approach to Industrial Software Engineering) (RAISE 1992). Yet no prior knowledge of the formalism is needed as modelling techniques are simple and fully explained.

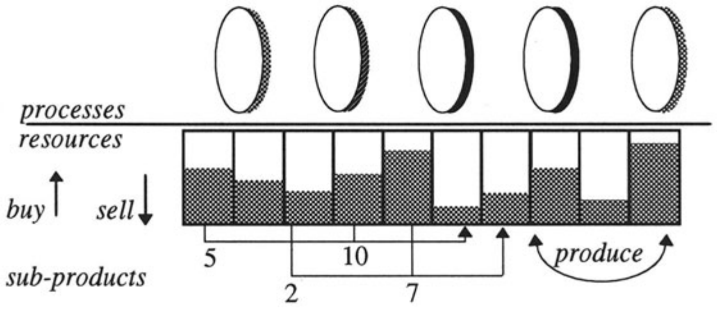

Figure 1 Modelling an enterprise, a unifying perspective

2.1 Products

Suppose there is a number of products we may be interested in. All such products are values of the abstract type `Product` (a type is a collection of values together with operations upon them). We are not interested in the detailed nature of products but in some attributes that will be important for modelling of the enterprise. The first relates to the storing requirements for items of the product, assumed to be represented numerically: function `store` from products to natural numbers. The second is a map (a partial function) from products to subproducts and their quantities: how many items are needed to assemble a single item of the product. This represents a bill of products.

type
  Product
value
  store: Product $\to$ **Nat**,
  bill: Product $\xrightarrow[m]{}$ (Product $\xrightarrow[m]{}$ **Nat**)

Not all such maps can reasonably represent a bill of products. We disallow products which are sub-products of themselves or subproducts which are not in the bill. The required properties are given by the axiom below, in terms of the predicate ‘is subproduct’; **dom** represents domain of a map (a set of arguments) and **pre** a pre-condition for a partial function.

```
value
  issub: Product × Product -~-> Bool
  issub(q,p) ≡ q ∈ dom bill(p) ∨
    (∃ r:Product • r ∈ dom bill ∧ issub(r,p) ∧ issub(q,r)) pre p ∈ dom bill
axiom ∀ p, q : Product •
  p ∈ dom bill ⇒ (~issub(p,p) ∧ (issub(q,p) ⇒ q ∈ dom bill))
```

2.2 Resources

We selected to model three kinds of resources: stock represents the number of items of each product available in the warehouse; space represents maximum capacity of the warehouse, ‘measured’ by summing up function `store` for all products and stocks (`storeall`); production capacity is a map from products to natural numbers, representing products which can be assembled on the shopfloor and how many can be processed within an agreed time.

```
type
  Space = Nat,
  Stock = Product -m-> Nat,
  Trans = Product -m-> Nat
value
  storeall: Stock → Nat
  storeall(s) ≡ sum ({s(p) * store(p) | p:Product • p ∈ dom s})
```

We represent resources as a record of the stock, space and production capacity: the stock must not exceed the space or include products not in the bill, and production must not apply to products not included, or their sub-products not included, in the stocks. A record is like a Cartesian product with functions for each field to extract and modify values. `Resources` is a sub-type of `Res`:

```
type
 Res :: stock : Stock  space : Space  trans : Trans,
 Resources = {| r:Res • iswf(r) |}
value
 iswf: Res → Bool
 iswf(r) ≡ dom stock(r) ⊆ dom bill ∧ storeall(stock(r)) ≤ space(r) ∧
   (∀p,q:Product • p ∈ dom trans(r) ⇒
     p ∈ dom stock(r) ∧ q ∈ dom bill(p) ⇒ q ∈ dom stock(r))
```

2.3 Activities

Operating on the resources are functions for buying, selling and producing products, and functions for expanding/reducing production and storing capacities. We only consider functions for stocks.

```
value
  buy, sell, prod : Product × Nat × Resources ⥲ Resources
```

Buying increments the stock, provided there is enough space. Selling decrements the stock, provided there is enough items.

```
axiom forall p:Product, n:Nat, r:Resources •
  stock(buy(p,n,r))(p) ≡ stock(r)(p) + n
    pre p ∈ dom stock(r) ∧ storeall(stock(r)) + n * store(p) < space(r),
  stock(sell(p,n,r))(p) ≡ stock(r)(p) – n pre p ∈ dom stock(r) ∧ stock(r)(p) ≥ n
```

Production decrements stocks for all immediate subproducts and increments the stock for the product. It depends on the availability of stock, space and production capacity.

```
axiom forall p,q:Product, n:Nat, r:Resources •
  stock(prod(p,n,r))(p) ≡ stock(r)(p) + n pre p ∈ dom trans(r) ∧
    n ≤ trans(r)(p) ∧ storeall(stock(r)) + n * store(p) < space(r),
  stock(prod(p,n,r))(q) ≡ stock(r)(q) – n * bill(p)(q) pre q ∈ dom bill(p) ∧
    stock(r)(q) ≥ n * bill(p)(q) ∧ p ∈ dom trans(r) ∧
    n ≤ trans(r)(p) ∧ storeall(stock(r)) + n * store(p) < space(r)
```

2.4 Processes

Deciding which activities should be carried out and in which order is the task of a process. A process is like a "program" which executes on the enterprise resources. The process can test the state of resources, e.g. if the stock is greater than given number, if the space can accommodate given number of items . . . Or involve a human decision, the outcome of which cannot be resolved based on the resources alone. Tests can also apply propositional constants `tt` (true), `ff` (false) and connectives `and` and `or`. The type of tests is a union type, which specifies alternative ways to build tests.

```
type
  Test == tt | ff | human |
    not(Test) | and(Test,Test) | or(Test,Test) |
    isempty(Product) | ismore(Product,Nat) | isspace(Product,Nat) | _
```

A process is one of resource-consuming activities: buying, selling or producing, and expanding/reducing space or production capacities. A process can be halted, a sequential or conditional execution of two processes (depending on the outcome of a test), an alternative execution which lets the environment decide on the execution of one of processes, or an execution which repeats a process as long as given test remains true. Processes are defined below.

```
type
 Proc == halt | space(Int) | trans(Int,Product) |
   buy(Nat,Product) | sell(Nat,Product) | prod(Nat,Product) |
   seq(Proc,Proc) | alt(Proc,Proc) | test(Test,Proc,Proc) | loop(Test,Proc) | _
```

2.5 Enterprise

The enterprise executes concurrently a set of processes, each given a unique identifier. To resolve competion for shared resources we assign to every process a natural number: the higher the number the higher the importance. The resources, the set of processes and priorities together "define" the enterprise.

```
type
 Pid,
 Ent:: res: Resources
   man: Pid →ₘ Nat
   beh: Pid →ₘ Proc
```

This representation is subject to some restrictions on: the names of processes (all must receive corresponding priorities) and products mentioned inside processes, which all must have their stocks present. **rng** represents the range of a map and `prods` returns the set of products inside the description of a process.

```
type
 Enterprise = {|e:Ent • iswf(e)|}
value
 iswf: Ent → Bool
 iswf(e) ≡ dom man(e) = dom beh(e) ∧
   (∀ p:Proc • p ∈ rng beh(e) ⇒ prods(p) ⊆ dom stock(res(e)))
```

This provides a general 'structure' for abstract description of enterprises. Concrete instances can be defined as values of this type. Consider a small example. Suppose we have products `p(i)` where `i` is a natural number and `p(i)` is the only sub-product of `p(i+1)` with quantity `i`, for all `i`. Suppose we have enterprises `e(i)`, each holding stock for `p(i)` and `p(i+1)`, and able to produce `p(i+1)` only. Each `e(i)` consists of one process `s(i)` which tries to repeatedly sell one item of `p(i+1)`. If the stock is empty, it tries to produce this item from `p(i)`, if the stock for `p(i)` is too low, it tries to buy `p(i)` first.

```
value
  s: Nat → Proc,
  p: Nat → Product,
  e: Nat → Enterprise
axiom ∀i:Nat •
  p(i) ∈ dom bill ∧ bill(p(0)) = [] ∧ bill(p(i+1)) = [p(i) ↦ I],
  dom trans(res(e(i))) ≡ {p(i+1)}, dom stock(res(e(i))) ≡ {p(i),p(i+1)},
  rng beh(e(i)) = {s(i)},
  s(i) ≡ let t = not(isempty(p(i+1))),
             r = test(ismore(p(i),i),prod(1,p(i+1)),buy(i,p(i)))
         in loop(tt,test(t,sell(1,p(i+1)),r)) end
```

3 MODEL COMPOSITION

We are now going to consider organisation of many enterprises cooperating or competing (or both) with each other. We study and interpret two ways to put enterprises together, by operations on their models (Figure 2): union and product. The first adds up all production capacities and processes in both enterprises, the second extracts those processes in both enterprises which utilise local as well as remote (owned by another enterprise) resources. We interpret them to represent an extended and a virtual enterprise respectively.

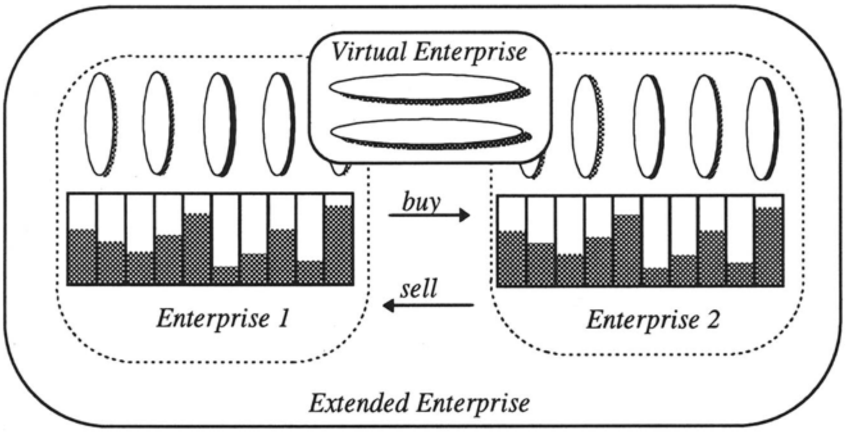

Figure 2 Model composition: the extended and the virtual enterprise.

3.1 The Extended Enterprise

The extended enterprise is an enterprise which consists of a set of enterprises (an industry), their internal resources, processes and possibility for exchange of products between them. The following function represents this exchange between two enterprises disregarding the running of processes in each of them.

```
type
 Id,
 Industry = Id →m Enterprise
value
 exchange: Id x Id x Product x Nat x Industry ~→ Industry
 exchange(s,c,p,n,i) ≡ i † [s ↦ sell(p,n,i(s)), c ↦ buy(p,n,i(c))] pre
  {s,c} ⊆ dom i ∧ stock(res(i(s)))(p)  ≥ n ∧
  p ∈ dom stock(res(i(s))) ∩ dom stock(res(i(c))) ∧
  storeall(stock(res(i(c)))) + n * store(p) ≤ space(res(i(c)))
```

On outside, an extended enterprise is like the usual enterprise in Section 2. There is no need to actually model the extended enterprise separately, only to show how to obtain it from the collection of enterprises. This is done by summing up all resources, processes and priorities:

```
value
 abstraction: Industry ~→ Enterprise
 abstraction(i) as ee post
  beh(ee) = mapunion({beh(e) | e:Enterprise • e ∈ rng i}) ∧
  man(ee) = mapunion({man(e) | e:Enterprise • e ∈ rng i}) ∧
  stock(res(ee)) = mapsum({stock(res(e)) | e:Enterprise • e ∈ rng i}) ∧
  trans(res(ee))  = mapsum({trans(res(e))  | e:Enterprise • e ∈ rng i}) ∧
  space(res(ee)) = numsum({space(res(e)) | e:Enterprise • e ∈ rng i})
```

We also need to justify that this abstraction was not too generous in removing internal details. The usual way is to show that there exists a correspondence between operations on the abstracted model and on components of the concrete model, so that they in some sense "simulate" each other. This is done by showing that the diagrams below commute (Janowski and Acebedo 1996).

$$\begin{tikzcd}
\mathit{ind} \arrow[r, "\mathit{abstraction}"] \arrow[d, "\mathit{buy/sell}"'] & \mathit{ext} \arrow[d, "\mathit{buy/sell}"] \\
\mathit{ind'} \arrow[r, "\mathit{abstraction}"] & \mathit{ext'}
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathit{ind} \arrow[r, "\mathit{abstraction}"] \arrow[d, dotted, "\mathit{exchange}\ \mathit{prod}"'] & \mathit{ext} \arrow[d, "\mathit{prod}"] \\
\mathit{ind'} \arrow[r, "\mathit{abstraction}"] & \mathit{ext'}
\end{tikzcd}$$

3.2 The Virtual Enterprise

Section 3.1 showed that any collection of enterprises may be reduced into a single enterprise. The 'inverse' in a sense doesn't hold: there is more capacity inside the extended enterprise that we could possibly exploit in terms of individual enterprises, e.g. to write a process which uses the resources of all of them. This section opens up this possibility. We revisit definitions of tests and processes to allow them to access local as well as remote resources (activities refer to the enterprise id), and to allow for exchange of products.

```
type
  Test' == tt | ff | human |
    not(Test') | and(Test',Test') | or(Test',Test') |
    isempty(Product,Id) | ismore(Product,Nat,Id) | _ ,
  Proc' == halt |
    exchange(Nat,Product,Id,Id) |
    space(Int,Id) | trans(Int,Product,Id) |
    buy(Nat,Product,Id) | sell(Nat,Product,Id) | prod(Nat,Product,Id) |
    seq(Proc',Proc') | alt(Proc',Proc') | test(Test',Proc',Proc') | _
```

We refine the enterprise accordingly into `Enterprise'`. Unlike an extended enterprise, a virtual enterprise is a set of processes only, which operate upon the resources of not only its own but also other enterprises. Definition is below. We also provide a function which extracts a virtual enterprise from given set of enterprises, provided no process refers to the enterprise outside the set. `eids` returns all enterprise identifiers present within a process description.

```
type
  VirtualEnt = Proc'-set,
  Industry' = Id $\underset{m}{\rightarrow}$ Enterprise'
value
  virtual: Industry' $\xrightarrow{\sim}$ VirtualEnt
  virtual(i) ≡
    { p | p:Proc' • ∃ e:Id • e ∈ dom i ∧ p ∈ rng beh(i(e)) ∧ eids(p) ⊃ {e} }
    pre (∀e,e': Id, p:Pid • e ∈ dom i ∧
         p ∈ dom beh(i(e)) ∧ e' ∈ eids(beh(i(e))(p)) ⇒ e' ∈ dom i )
```

4 CONCLUSIONS

There is a growing demand for tools (Christiansen 1997), methods (Williams 1994), languages and standards (Clements 1997) to model, analyse, build and re-build manufacturing enterprises as engineering artifacts. This, however, requires formalisation of the basic concepts which underline the 'definition' of the enterprise and its derivatives (extended/virtual enterprise).

We presented a simple model for the class of enterprises, building on the concepts of products, resources, activities upon resources, processes which govern execution of activities, leading to the enterprise itself: a set of processes, with priorities, executed concurrently on the shared resources. We presented two ways to compose such models, corresponding to the 'union' and 'product', and interpreted respectively as an extended and a virtual enterprise.

Although we presented the models formally, we only explained informally their semantics. Formal semantics of the models presented here is subject of a companion paper (Janowski et al. 1998). This can help to study ways of reasoning (symbolically) about the enterprise, its evolutions and derivatives, and provide a formal underpinning for building tools for symbolic prediction and optimisation of business operations in manufacturing.

REFERENCES

Bernus P. and Nemes, L. (1995). Enterprise Integration - engineering tools for designing enterprises, *in* P. Bernus and L. Nemes (eds), *Modelling and methodologies for Enterprise Integration*, IFIP, Chapman and Hall.

Browne, J.(1995). The Extended Enterprise - Manufacturing and The Value Chain, *in* L. M. Camarinha-Matos and H. Afsarmanesh (eds), *BASYS95*, Chapman and Hall.

Christiansen, T. (1997). Integrated tools in support of enterprise modelling and analysis, *Previous Workshop 5-Intl. Conf. on Enterprise Integration Modelling Technology.*

Clements, P. (1997). Standards support for the integration and interoperation of the virtual enterprise, *Previous WS 2-Intl. Conf. on Enterprise Integration Modelling Technology.*

Davidow, W. and Malone, M. (1992). *The Virtual Corporation: Structuring and Revitalizing the Corporation for the 21st Century*, Harper Collins, New York.

RAISE, The RAISE Language Group (1992). *The RAISE Specification Language*, Prentice Hall.

Janowski, T. and Acebedo, C. (1996). Virtual Enterprise: On Refinement Towards an ODP Architecture, *Technical Report 69*, UNU/IIST.

Janowski, T., Zheng, H. and Giménez Lugo, G. (1998), Market-Driven Symbolic Execution of Models of Manufacturing Enterprises, *Technical Report 137*, UNU/IIST.

Hardwick, M., Rando, T., Spooner, D. and Morris, K. (1997). Data Protocols for the Industrial Virtual Enterprise, *IEEE Journal of Internet Computing*, **1**.

Pant, S. and Hsu, C. (1996). Bussiness On The Web: Strategies and Economics, *Fifth International WWW Conference*, Paris, France.

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

Williams, T. (1994) The Purdue Enterprise reference Architecture, *Computers in Industry* **24**(2): 141-158.