GENERIC BUSINESS MODELLING FRAMEWORK

Christopher John Hogger and Min Li
Department of Computing, Imperial College London, South Kensington Campus, London SW7 2AZ, United Kingdom

Keywords: Business models, business processes, enterprise management, requirements, constraint evaluation.

Abstract: We present a position paper setting out the essentials of a new declarative framework named *GBMF* intended for modelling the higher-level aspects of business. It is based upon logic programming including, where appropriate, finite-domain constraints. Business plans, processes, entity constraints, assets and business rules are representable in *GBMF* using an economical repertoire of primitive constructs and without requiring overly-burdensome programming effort. The framework, which has been fully implemented, has been applied so far to small-scale business exemplars. Our more general future aim, however, will be to demonstrate the framework's generic character by providing precise semantic mappings between it and other business modelling frameworks that rely upon specialized languages and engines.

1 INTRODUCTION

The modelling of businesses, their methods and their processes has been pursued within many frameworks and perspectives. Our work aims to establish a generic framework serving as an abstract, but implementable, representation of core features of the more concrete and established frameworks now existing, and so provide a common semantic basis facilitating their comparison and inter-translation.

Our generic business modelling framework, *GBMF*, is built upon the general notion of activities operating upon *entities*. The latter may be variously physical (manufacturing parts, resources), electronic (databases, emails, websites), financial (accounts, budgets), human or any typical business entities. Activities performed upon them are composed from atomic *basic actions* organized into *action sets*, which are in turn organized into larger programmatic hierarchies called *business plans*. A plan might embody the actions entailed in a production process from inception to delivery, with attendant impacts upon financial and temporal aspects of the business. Such a plan may be applied on multiple occasions, possible concurrently. *GBMF* therefore treats a plan as a template capable of spawning distinct instances called *processes*, each acting upon its own vector of business entities. The instigations and progressions of processes are governed by *business process rules*, whilst the internal relationships between their entities are governed by the underlying procedures that define the basic actions and, (if required) by separately given *business entity constraints*. Many entities in a process will have a transient existence, being only intermediates for creating the eventual deliverables of that process. When the process terminates, these intermediates have no further significance and are discarded. Those entities to which this does not apply are the deliverables that must survive, referred to as *business assets*. Thus the macroscopic behaviour of an executing *GBMF* instance is the *transformation of an asset space*, as various processes are spawned, possibly exploiting pre-existing assets and, usually, creating new assets.

Sections 2 , 3 and 4 introduce plans, processes and business entity constraints, respectively. The notion of assets is introduced in Section 5, and business process rules in Section 6. Related work is compared with *GBMF* in Section 7 whilst Section 8 states conclusions and future intentions.

2 *GBMF* PLANS

GBMF represents each basic action as a term of the form *Action-name(Ontvars)* in which *Ontvars* is a vector of ontological variables each representing an entity. By convention, the last argument in this vector always represents the time interval over which the action is performed. Each basic action *A* appears within an *action declaration* whose simplest

John Hogger C. and Li M. (2007).
GENERIC BUSINESS MODELLING FRAMEWORK.
In *Proceedings of the Ninth International Conference on Enterprise Information Systems - ISAS*, pages 422-427
DOI: 10.5220/0002407704220427
Copyright © SciTePress

form is *action(S, I, A)* where *S* names an action set and *I* is a position index for *A* within *S*.

```
action(dohire, 1, hire(request, tool, hiring, t4)).
action(dohire, 2, hireadmincost(expense, t5)).
action(dohire, 3, recordexpense).
action(dohire, 4, publish(hiring, nc, t6)).
```

Figure 1: The ‘dohire’ action set.

Figure 1 shows action declarations, indexed 1-4, expressing an entire action set named ‘dohire’. The general syntax of an action declaration is

action(S, I, X).
or *action(S, I, C, X)*
or *action(S, I, C, X1, X2)*

in which each of *X*, *X1*, *X2* is a basic action or an action set name and *C* is a predicate, over one or more ontological variables, expressing a condition or a decision. The first form above appears throughout in Figure 1, but at index 3 expresses that another action set, ‘recordexpense’, is to be performed unconditionally. The second form expresses that if *C* holds then *X* is performed, whilst the third form expresses that if *C* holds then *X1* is performed but otherwise *X2* is. Neither of these conditional forms is needed in ‘dohire’. The ‘dohire’ action set is implemented as the activity of satisfying a customer request (to hire a tool) by finding such a tool already in stock and thereby entering a hiring agreement, then looking up the administrative expense to the business of doing all this, then recording the expense (charging it to an account) and finally making the hiring ‘public’ in the sense of making it available as an asset for other processes to operate upon.

The names of action sets are chosen freely by the user (modeller). The names (e.g. ‘hire’ above) of basic actions are either freely user-defined or belong to a small set of system-defined names (e.g. ‘publish’ above) having fixed meanings in the framework.

The ability of one action set to invoke others, conditionally or otherwise, inherently organizes a complete *GBMF* model into a set of plans. A plan comprises a *root* action set - characterized by being invokable by no other - together with all those other action sets that it may invoke directly or (through those others) indirectly. This allows an action set to belong to more than one plan, if required, to economize on the use of common knowledge.

A plan *P* has an associated ontology *ont*(*P*) which contains the alphanumeric arguments in the user-defined basic actions in *P*. It also contains the arguments of system-defined basic actions in *P* that are known - from a separate framework dictionary - to be ontological variables. In the ‘publish’ action, for example, it is known that only its first and third arguments are of this kind So, if *P* uses the ‘dohire’ action set then, from Figure 1, *ont*(*P*) must include at least {request, tool, hiring, expense, t4, t5, t6}. The names of ontological variables are chosen freely.

Each action set has an associated *control declaration* which is either of the assertions control(seq) or control(con). This specifies whether the basic actions in the set are to be performed sequentially or concurrently. The former case uses the action declarations’ indices to determine the temporal order, whilst the latter case ignores them.

3 *GBMF* PROCESSES

A *GBMF process* is an executing instance of a plan. At any time in the animation of a model there may exist zero or more active instances of each plan, at various stages in their executions. The factors governing process initiation, progression and termination will be outlined in the next section.

A process is denoted P_i where *P* is a plan and *i* is a unique instance identifier. P_i has its own binding environment $\beta(P_i)$ containing a pair (*V*, *Val*) for each $V \in ont(P)$ signifying that *V* is *bound* to the value *Val*. When P_i is initiated, $\beta(P_i)$ consists of null bindings, i.e. P_i’s variables are uninstantiated.

As P_i executes, its variables become instantiated by various ways - clock-binding, action performance and constraint evaluation. Clock-binding instantiates the start (*s*) or the end (*e*) in the time-interval value (*s*, *e*) of the temporal variable in a basic action *A* when *A* is about to be performed or has just been performed. It instantiates *s* or *e* with the current time as given by the clock in the model’s execution manager. Performing *A* entails consulting an *Action Knowledge Base* (*AKB*) containing, for each basic action type, an associated procedure. If *A* is user-defined then its procedure will have been supplied in the *AKB* by the modeller. The primary effect of executing the procedure is to update $\beta(P_i)$. A secondary effect, relating to asset management, will be described presently, as will constraint evaluation.

Figure 2 shows the items described so far in a run of a given model. On the left are static resources including the plans - defined entirely by action declarations and control declarations - and the *AKB*. To the right is the dynamic component consisting of a pool of active (unterminated) processes, their binding environments β and their activation records α representing their states of progress.

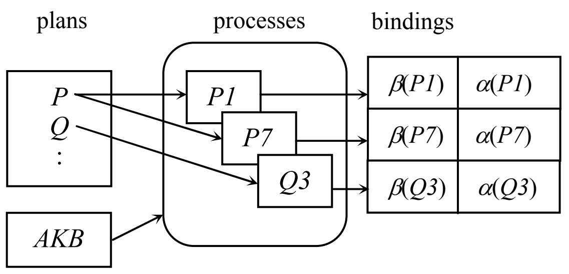

Figure 2: Active process pool.

4 ENTITY CONSTRAINTS

Business entity constraints in *GBMF* are required relations over ontological variables. A *constraint declaration* takes the form *constraint*(*C*) where *C* is some predicate having a supporting *constraint definition* capable of evaluating it - for example:

```
constraint(hiring_span(t4, t6)).
hiring_span((S, _), (_, E))  if  (E - S) < 6.
```

Referring to the example in Figure 1, this constraint requires the total duration of the activity entailed in the ‘dohire’ action set to be less than 6 time units. In the operating model, this constraint applies to every process that potentially binds (incarnations of) t4 and t6. In the tool-hire model these two variables jointly occur in a single plan, ‘toolhire’ but, more generally, constraints may be imposed on processes spawned from different plans. For example, the constraint con25 in the declaration

```
constraint(con25(v3/plan4, v5/plan1)).
```

constrains the relation between v3 in *ont*(plan4) and v5 in *ont*(plan1) and is applied to every process pair (π_i, π_j) such that π_i and π_j are instances of plan4 and plan1, respectively.

In the running model, constraints are evaluated by a separate *constraint manager* acting in concert with the execution manager responsible for spawning and advancing processes. In the current implementation of *GBMF* they are tested each time an action is performed. More generally, however, constraints may be tested at any time. Their definitions may if required, be *FD-constraint logic programs* which effectively bind selected variables to finite domains of feasible values. The domains become successively narrowed by the various effects of clock-binding and/or action-performance, and for as long as these domains remain non-empty the constraints remain satisfiable. Constraints of this kind can be evaluated even prior to such events, enabling forward planning in respect of such things as resource allocation, budgeting and scheduling.

Currently, the satisfiability of constraints is not enforced in *GBMF* - instead, the implementation just reports constraint violation if and when it occurs. However, constraints can facilitate collaboration between the various agents within a business. In a simpler model (Hogger, 2004) precursive to *GBMF*, plans and constraints are fluidly associated with *agent roles* and can be consistency-tested at role-formation time. If constraints become subsequently violated as concrete plan instances proceed, failure analysis can identify the agents responsible, who can then confer upon appropriate remedies such as constraint revision. In due course we hope to add these features to *GBMF* and further combine them with collaborative treatment of the model’s *business process rules*, discussed in Section 6.

5 ASSETS

By default, the termination of a process would leave no trace of its prior existence, since its bindings are then automatically garbage-collected. Instances of relations between its ontological variables would have been constructed or verified by the effects of actions and constraints, but would not survive to the lasting benefit of the modelled business as a whole.

In order to enable processes to manipulate business entities of greater permanency, *GBMF* allows the modeller to declare, for any basic action type, that some of its arguments denote durative assets. Concretely, such an asset is a value *Val* - in general a compound structure conforming to a schema declared in the *AKB* for a particular asset *type* - tagged with a unique identifier, a type, a status and an origin. When it comes to exist, during the running of the model, it will do so within an *asset space* that is adjoint to the process pool. The *status* of the asset, at any instant, is either public, i.e. it is available for use by any process, or is owned, i.e. it belongs (at this moment) to a particular process. Its *origin* identifies the process π_o from which it originated together with the name of the variable *V* that became bound to *Val* by executing π_o and the time interval of the particular action in π_0 that created the asset. Equivalently, the asset can be seen as a tagged copy of the binding (V, Val) that was constructed in the environment $\beta(\pi_0)$. Assets may serve many purposes relating to process inter-dependence, including message-passing.

Figure 3 shows an asset created in a run of a tool-hire business model, of which the action set in Figure 1 is a small fragment. This asset, of type 'administration:toolcatalogue', is a list of tooltypes. Its originating process was an instance '270' of a 'stock' plan which, in the interval from s=303 to e=304, performed an action that bound the variable 'updatedinventory' to this list. Since then, the asset has passed to the ownership of another process, namely instance '327' of a 'toolhire' plan.

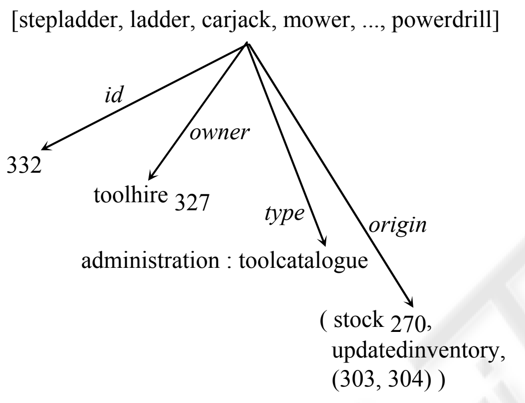
[stepladder, ladder, carjack, mower, ..., powerdrill]
id
owner
type
origin
332
toolhire 327
administration : toolcatalogue
(stock 270, updatedinventory, (303, 304))

Figure 3: Concrete asset.

Each process π has an associated *asset register*, denoted *areg*(π) and initially empty, which records the assets currently owned by π. Each entry is a pair (*id*, V) where *id* is the asset's unique (and invariant) identifier and V is a variable in π's ontology.

Besides the action-defining procedures, the *AKB* contains *asset declarations* specifying any asset-handling entailed in each action type. For some basic action A, such declarations may express that some of A's variables variously denote *pre-assets*, *post-assets* or *consumed-assets*. However, not all actions need involve handling assets.

When a process π starts to perform A, π is expected to own an asset for each of A's pre-asset variables. One way this can arise is by π having already performed system-defined actions available for acquiring or copying public assets. However, if a pre-asset variable V in A is not already associated with a π-owned asset then the asset space is searched for a public one whose type matches the schema for V given in the *AKB* and whose value is compatible with the binding of V in $\beta(\pi)$. If one is found then a copy of it is assigned to π's ownership and is duly associated with V in *areg*(π), otherwise A *suspends* until an appropriate asset becomes available.

When a process π is about to finish performing A, the values to which A's post-asset variables (if any) are bound in $\beta(\pi)$ are used to construct new assets belonging to π and the associations of these assets with these variables are recorded in *areg*(π).

Further, when a process π finishes performing A, the assets with which A's consumed-asset variables (if any) are associated in *areg*(π) are deleted from the asset space and from *areg*(π) itself.

For example, the tool-hire model's *AKB* contains these asset declarations for the 'hire' action type:

```
pre_assets(hire(Request, Tool, _, _), [Request, Tool]).
post_assets(hire(_, _, Hiring, _), [Hiring]).
consumed_assets(hire(Request, Tool, _, _),
                        [Request, Tool]).
```

They require that when a 'hire' action begins it must have pre-assets associated with variables Request and Tool, and when the action ends it must convert its value for Hiring to a post-asset and moreover consume (delete) the assets associated with Request and Tool. The latter is just a nuance allowing one to exercise fine control over asset lifetimes. It is not essential because, when a process π terminates, assets that it owns are in any case garbage-collected.

This last remark implies that if any of π's assets are required to survive π's termination then π must beforehand make them public. To do so it can perform a system-defined basic action 'publish'. This is seen in Figure 1 where the asset associated with Hiring is made public. When a complete run of the model terminates, what survives is just the set of public assets still remaining in the asset space. These are the only observable deliverables of the business.

6 BUSINESS PROCESS RULES

The plans and the *AKB*'s asset declarations in a model induce a directed *plan-asset dependency graph* whose nodes are the model's plan names. Each edge directed from P to Q is labeled by the set of those variables in *ont*(P) that P potentially delivers as assets to Q. Although these dependences are logical consequences of the model and can be mechanically compiled from it, it serves the modeller's interest to assert them explicitly in a separate component of the model called the *Business Process Rulebase* (*BPR*). This is because they contribute to the formulation of *business process rules* regulating process creation and behaviour, though not all such rules refer to asset handling.

Figure 4 outlines a small fraction of the graph for the tool-hire business model.

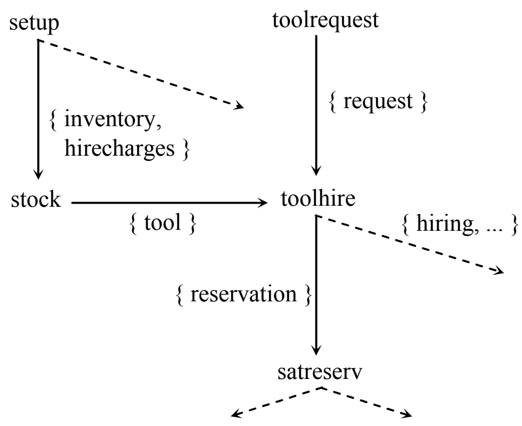

Figure 4: Plan-asset dependency graph.

Thus we see that a 'toolhire' process expects, at the least, to be served by a tool asset from a 'stock' process and a request asset from a (customer's) 'toolrequest' process, and to deliver a hiring asset or a reservation asset to yet other processes.

In the *BPR* each edge of the graph is declared by an assertion of the form

```
        plan_asset_dep(P, Assetvars, Q).
e.g.  plan_asset_dep(stock, [tool], toolhire).
```

Business process rules in the *BPR* may or may not exploit plan-asset dependences. Examples are:

```
  initiate_process(Q) if  Q \= setup,
      plan_asset_dep(P, Assetvars, Q),
      member(V, Assetvars),
      asset_space_has(V), origin(V, P), status(V, public),
      not_engaged(Q, V).
  initiate_process(setup) if  asset_space_lacks(inventory).
```

These declare that a process for any plan *Q*, other than one for 'setup', be initiated if the asset space contains a public asset originating from a variable *V* in *ont*(*P*), provided *V* contributes to the dependence of *Q* upon *P* and provided no other process for *Q* is already engaged to make use of that asset. On the other hand, a process for 'setup' - of which only one will ever be executed - should be initiated if no inventory (which it is the exclusive duty of 'setup' to create) has yet appeared in the asset space.

More generally, the *BPR*'s business process rules can express any requirements about the behaviour of the process pool if these are expressible as logical conditions over existing processes, their associated binding environments or activation records or the contents of the asset space. They are consulted by the model's execution manager to drive the model forwards and to ensure that each process is spawned to serve a declared purpose and that its subsequent behaviour satisfies any declared conditions.

7 RELATED WORK

Many recommendations exist as to the macroscopic concepts that need to be considered in business modelling. For (Fox, 1998) these concepts need to be based fundamentally on well-defined ontologies. For (Hamel, 2000) they include strategic resources, cost interfaces and value networks, for (Alt, 2001) they include mission, processes and revenues, whilst for (Affua, 2003) they include dynamics, taxonomy and value. There are clearly intersections and exclusions among these concept sets, but ascertaining this precisely is difficult since not all of them are given sufficiently formal anchorage.

Existing business models vary from the most abstract, e.g. very general axiomatizations, to mid-level ones exposing more detailed commitments to representation, logic and behaviour, down to concrete, context-specific implementations. *GBMF* lies in the middle level. The latter has been broadly characterized by (Chen-Burger, 2002) as expressing conditions and actions of business processes, relationships between them and constraints on the data they deal with. Other concept-focused examplars here include *ENTERPRISE* (Uschold, 1998) and *e3value* (Gordijn, 2003), the latter being compared in detail by (Gordijn, 2005) with *BMO* (*Business Modelling Ontology*) (Osterwalder, 2005).

Instances of *BMO*, implemented in XML and OWL, are enterprise-specific representations of, chiefly, offers, customer interfaces, infrastructure and - especially - the logic entailed in earning revenue. Unlike *GBMF* models, concrete instances of *BMO* do not operate under the control of an executable enterprise-independent superstructure.

Besides those frameworks that constrain the structure of business models, there are many others intended for facilitating their design and implementation. Early exemplars include the *BSDM Business System Development Method* (IBM, 1992), the *Ordit* organisational modelling method (Dobson, 1994) and the business process modelling methods *IDEF0* (NIST, 1993) and *PSL* (Schlenoff, 1997). The one most similar to, but more mature than, *GBMF* is the *Fundamental Business Process Modelling Language FBPML* (Chen-Burger, 2002 & 2004; Kuo, 2003), a sophisticated amalgamation and extension of features drawn from *PSL* and *IDEF3*. *FBPML* is declarative, using logic to describe features of, and relations over, business processes. It further includes tools for developing, testing and analysing models, and also an engine for eliciting workflow animations from them.

Though sharing similarities with *GBMF*'s basic representations of actions, preconditions, entities and of process logic and behaviour, *FBPML* is a special-purpose language requiring its own custom-built engines and tools, whereas *GBMF* models are written directly in the general-purpose Prolog language and so freely inherit all the representational and executional power of that formalism, besides the standardized and well-understood stable model semantics of normal-clause logic. An additional and significant difference is that *GBMF* can, as we have indicated earlier, exploit the expressiveness and computational power of finite-domain constraints.

8 CONCLUSIONS

We have outlined a declarative, context-independent and implementable framework for modelling aspects of business. Formulating this in logic programming gives the benefits of general-purpose expressiveness and well-understood execution regimes, so avoiding the need for a special-purpose engine supporting a specialized modelling language. Process plans, constraints and asset management are expressible transparently using a small range of basic constructs. Our main aim is to exploit the well-understood semantics of logic programs in a future programme of work intended to map other frameworks such as *FBPML* to *GBMF* and thus to establish the generic nature of the latter and to facilitate inter-framework comparison and translation as explored in, for example, the work of (Chen-Burger, 2001).

REFERENCES

Afuah, A., Tucci, C.L., 2003. *Internet business models and strategies*. McGraw Hill. Boston.

Alt, R., Zimmermann, H., 2001. Business models. In *EM-Electronic Markets 11(1)*.

Chen-Burger, Y-H., 2001. Knowledge sharing and inconsistency checking on multiple enterprise models. In *IJCAI'01, 17th International Joint Conference on Artificial Intelligence, Workshop on Knowledge Management and Organizational Memories*.

Chen-Burger, Y-H., Tate, A., Robertson, D., 2002. Enterprise Modelling: A declarative approach for FBPML. In *European Conference on Artificial Intelligence: Workshop on Knowledge Management and Organizational Memories*.

Chen-Burger, Y-H., Robertson, D., 2004. *Automating business modelling*. Book Series of Advanced Information and Knowledge Processing, Springer Verlag. Berlin.

Dobson, J.E., Blyth, A.J.C., Chudge, J., Strens, M.R., 1994. The ORDIT approach to organizational requirements. In *Requirements Engineering: Social and Technical Issues*. Academic Press. London.

Fox, M.S., Gruninger, M., 1998. Enterprise modeling. In *AI Magazine 19(3)*.

Gordijn, J., Akkermans, H., 2003. Value-based requirements engineering: exploring innovative e-commerce ideas. In *Requirements Engineering 8(2)*.

Gordijn, J., Osterwalder, A., Pigneur, Y., 2005. Comparing two business model ontologies for designing e-business models and value constellations. In *18th Bled e-Conference on e-Integration in Action.*

Hamel, G., 2000. *Leading the revolution*. Harvard Business School Press. Boston.

Hogger, C.J., Kriwaczek, F.R., 2004. *Constraint-guided enterprise portals*. In *ICEIS-2004*, *6th International Conference on Enterprise Information Systems.*

IBM, UK, 1992. *Business System Development Method: Business Mapping Part 1: Entities,* 2nd edition.

Kuo, H-L., Chen-Burger, Y-H., Robertson, D., 2003. Knowledge management using business process modeling and workflow techniques. In *IJCAI'03*, *18th International Joint Conference on Artificial Intelligence, Workshop on Knowledge Management and Organizational Memories*.

NIST - National Institute of Standards and Technology, 1993. *Integration Definition for Function Modelling (IDEF0)*. NIST. Gaithersburg.

Osterwalder, A., Pigneur, Y., Tucci, C.L., 2005. Clarifying business models: origins, present, and future of the concept. In *Communications of the Association for Information Systems, 16.*

Schlenoff, C., Knutilla, A., Ray, S., 1997. In *Proceedings of the Process Specification Language (PSL) Roundtable.* NIST. Gaithersburg.

Uschold, M., King, M, Moralee, S., Yannis Zorgios, Y., 1998. The enterprise ontology, In *Knowledge Engineering Review, 13.*