Policy Driven Business Performance Management

Jun-Jang Jeng, Henry Chang and Kumar Bhaskaran

IBM T.J. Watson Research Center
New York 10598, U.S.A.
{jjjeng,hychang,bha}@us.ibm.com

Abstract. Business performance management (BPM) has emerged as a critical discipline to enable enterprise to manage their business solutions in an on demand fashion. BPM applications promote an adaptive means by emphasizing the ability to monitor and control both business processes and IT events. However, most BPM processes and architectures are usually linear and rigid; and once done, will be very hard to change. Hence, it does not help enterprise to create adaptive monitoring and control applications for business solutions. There is an urgent need of adaptive BPM framework to be used as a platform of developing BPM applications. This paper presents a policy based BPM framework to help enterprise to achieve on demand monitoring and control framework for business solutions.

1 Introduction

Business performance management (BPM) has emerged as a critical discipline to enable enterprise to manage their business solutions in an on demand fashion. BPM applications promote an adaptive strategy by emphasizing the ability to monitor and control both business processes and IT events. By coordinating the business and IT events within an integrated framework, decision makers can quickly and efficiently align IT and human resources based on the current business climate and overall market conditions. Business executives can leverage the results of core business process execution to speed business transformation, and IT executives can leverage business views of the IT infrastructure to recommend IT-specific actions that can drive competitive advantage.

However, most BPM processes and architectures are usually linear and rigid; and once done, will be very hard to change. To change the requirements of BPM is sometimes like building a completely new application, which costs time and money. Some enterprises attempt to increase the flexibility and agility of business by introducing dynamic workflows and intelligent rules. However, this kind of **systems** is hard to be modeled, deployed and maintained. In the BPM domain, business analytics are commonly incorporated in business monitoring and management systems in order to understand the business operations in a deeper sense,. Nevertheless, most functions provided business analytics are performed in batch mode – unable to resolve business situations and exceptions in a timely fashion. How to run analytics in a continuous sense is a challenge. In general, it is extremely difficult to model, integrate and deploy

monitoring & control capabilities into larger scale business solutions such as supply chain management.

This paper presents a policy based BPM framework to address the above issues. A BPM system is a system for sensing environmental stimulus, interpreting perceived data, adjudicating the data to be business situations, and making decisions about how to respond the situations. A BPM system takes monitored data from target business solutions (e.g. business events), invokes BPM services and renders actions back to target business solutions. In general, there are five representative categories of services in a BPM system: Sense, Detect, Analyze, Decide and Effect. "Sense" is the stage when a BPM system interacts with business solutions and provides data extraction, transformation, and loading capabilities for the sake of preparing qualified data that is to be further monitored and analyzed. "Detect" is the stage of detecting business situations and/or exception occurring in the business solutions. "Analyze" is the stage when a BPM system performs business analytics such as risk-based analysis of resolving business exceptions. "Decide" is the stage when a decision maker will make decision about what to respond to business situations. A decision maker can be either human or software agent. "Effect" is the stage when a BPM system carries out actions for the purpose of enforcing the decisions made by decision makers. Actions can be of many forms. The simplest kind of action is alerting interested parties about the decisions. More complicated ones may be involved sophisticated process invocation.

As a motivating example for this paper, we want to show a BPM system for managing business solution that we built for some microelectronics manufacturer [1]. It comprises a suite of event-driven, decision management applications that enable proactive management of business disruptions in real time. The system's ability to identify potential out of tolerance situations, whether to unexpected fluctuations in supply and demand, or emerging customer, partner, and supplier needs, is enabled by analytical exception detection agents. These agents utilize standardized or configurable measurements to observe business events; for example to ensure that enterprise revenue goals are being accomplished. The BPM policies are managed pro-actively. Alert messages inform business process owners in advance if a new trend is emerging and actions must be taken. Finally, this system provides a suite of domain-dependent optimization, performance prediction, and risk assessment agents that make exception management even more effective. The agents adopt existing cost structures and business process flexibility, and recommend optimized business policies and actions that drive business performance to higher levels of productivity, efficiency, and financial predictability. The following scenario illustrates a scenario how the business line manager utilizes the BPM system.

- The BPM system receives *events* from various source systems from the supply chain. Some of these events impact the inventory levels or revenue metrics for the manufactured modules (such as "*order placed*" or "*order cancelled*" events). The BPM system continuously updates the actual revenue, the revenue outlook and inventory levels.
- Whether the progression of the accrued revenue is normal or below target is determined by the BPM system using a wineglass model [2]. In the case where the revenue is below target, the system automatically detects such a *situation* and issues an

alert showing the current sales quantities of some selected saleable part numbers in the *n*th week are out of their bands.

- The BPM system recommends adjusting the planned demand quantities and safety stock requirements for the *n*th week. As next step, it invokes a demand planning module and inventory planning module to analyse demand quantities and safety stock requirements for the *n*th week.
- It further recommends altering the daily build plan in order to optimally match new daily demand statements, thus high serviceability, and minimize manufacturing and inventory costs. By doing so, it also shows the effects and risks of all suggested alternatives for changing the build plan.
- Finally, the business line manager looks at the suggestions of the BPM system and makes a final decision for improving the build plan.
- The BPM system immediately revises the actual build plan in the ERP system (*action*) and continues the monitoring of the performance indicators with the updated build plan.

This paper is organized as follows. Section 1 introduced the BPM concept and a motivating example. Section 2 describes the concepts and lifecycle of BPM policies. Section 3 presents the policy-driven architecture for BPM. The related work is given in Section 5. Finally, Section 6 concludes this paper and discusses the future endeavour.

2 Defining BPM Policies

A BPM system is meant to be a platform for adaptive enterprise information systems in that the system behavior can be altered without modifying the mechanisms of the system itself. A BPM policy aims to govern and constrain the behavior of the BPM net and its constituent services. It usually provides policy rules for how the BPM system should behave in response to emergent situations [3]. As an example, a policy of supply chain inventory may impose limits on the range of inventory levels for the manufacturing process based upon the revenue target of the enterprise. Relevant policies can be devised and applied to different aspects of business solutions. Examples include role-based authorization to manage target business solutions and resources, the scope of managed business solutions and resources, and service-level agreements. Every BPM policy has its own lifecycle. The lifecycle of a policy consists of six basic life-stages as shown in Figure 1. They are: policy definition, policy activation, policy passivation, policy deployment and configuration, policy enforcement and policy termination.

Policy Definition is the phase that a policy is created, browsed and validated. Corresponding definitional tools such as editor, browsers and policy verifiers can be used by business analysts to input the different policies that are to be effective in the BPM system. *Policy Deployment & Configuration* configures and deploys a policy into target system and configures the system correspondingly. *Policy Enforcement* is the

stage when a policy is being enforced to govern and constrain the behavior of target systems. *Policy Activation* is the phase when a policy is loaded into target system and waiting for further execution. *Policy Passivation* is the phase when a policy is put to persistent storage without any active activity. *Policy Termination* is the phase when a policy ceases to exist in the system.

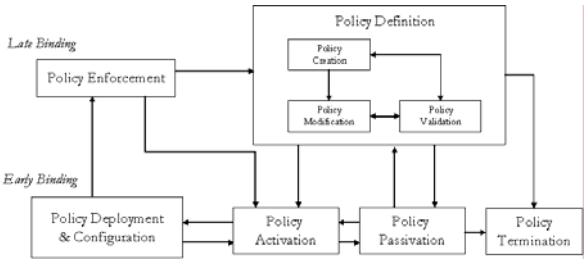

Fig. 1. Policy Lifecycle.

Potentially, a policy can be bound to BPM services at two points of its lifecycle: (1) policy deployment & configuration: this type of binding is called early binding between policy and mechanism since it is realized at the build time; and (2) policy enforcement: this type of binding is, on the other hand, called late binding between policy and mechanism since this binding is realized at the run time when policy is being executed.

The BPM policies are specified using Ponder-like expressions [4][5] as follows. In this syntax, every word in bold is a token in the language and optional elements are specified with square brackets [ ]. The policy with name "policyName" will be triggered when the events specified in "event-specification" are generated and captured by the BPM system. The event can be primitive event and compound event what composed from primitive event using event operator [6]. The keyword *subject* refers to the service that will act as the policy enforcer, and the *scope* phrase indicates the scope in that this policy will be applied. The "do-when" pattern signifies the actions to be enforced based on the pre-defined constraints.

```
policy policyName[(<type>argName[,<type> argName]*)]
                    on event-specification;
subject [<type >] domain-Scope-Expression;
[scope [<type >] domain-Scope-Expression;]
do action-list;
[when constraint-Expression ;]
```

The following segment shows the policy of detecting the out-of-bound revenue situation based on (a) given upper- and lower-bounds; and (b) predicted revenue performance. A metric event carrying the context object of the MDBPM system (noted as MDBPMContext) acts as an input to this policy. Some of the data referred by this policy are parameterized as input parameters: (1) *upperBound* is the upper bound of

the revenue performance; (2) lower*Bound* is the lower bound of the revenue performance; (3) *ActionPlanningService* indicates the service to receive the detected situation; (4) *LOBManager* is the manager who will get notified when the situation is eventually detected.

```
policy senseOutOfBoundRevenueSituation(
            int upperBound,
              int lowerBound,
              ActionPlanningService aps,
              LOBManager lob)
            on MetricEvent(MDBPMContext context);
 subject PolicyManager;          // the policy controller
target SituationDetectionService;//the policy enforcer
do {
     //  notify action planning service
     notify(aps, "OutOfBoundRevenueSituation",context);
     // notify LOB manager
     notify(lob, "OutOfBoundRevenueSituation",context);
}
// situation detection rule
when context.revenue > upperBound \/ context.revenue <
lowerBound ;
```

The following policy shows what needs to be actually done when the aforementioned situation occurs. This policy is triggered by a situation event carrying the MD context object *MDBPMContext*. The *do* clause defines an action by concatenating three other actions: (1) invoke the demand planning service to create a demand plan based on input situation object; (2) invoke the inventory planning service to create an inventory plan based on the demand plan; (3) notify the LOB manager about the recommended inventory plan. The execution strategy (as an input parameter) is DO_ALL_IN_SEQUENCE meaning every action indicated in *do* clause needs to be executed with indicated sequence.

```
policy respondOutOfBoundRevenueSituation(
     DemandPlanningService dps,
     InventoryPlanningService ips,
     LOBManager lob,
     ExecutionStraegy DO_ALL_IN_SEQUENCE)
     on SituationEvent(MDBPMContext context);
subject PolicyManager;
target ActionPlanningService;
do {
     // invoke demand planning service
     demandPlan = invoke(dps, demandPlan, context);
     // invoke inventory planning service
     inventoryPlan = invoke(ips, demandPlan, context);
     notify(lob, inventoryPlan, context); //notify LOB
manager
}
```

3 Policy Architecture

This section shows a realization of policy-driven BPM architecture. Two fundamental notions are presented here: *BPM ring* and *BPM net*.

BPM Rings

The BPM cycle is realized into BPM ring. A BPM ring represents a scalable mechanism of realizing real-time BPM capabilities at various levels of granularity (e.g. business organization, enterprise, value-net). A BPM ring consists of nodes and links. A BPM node is a basic service that enables transformation from input data to output data based on its capabilities and the pre-defined policies. A BPM link transmits data with specific types from one node to another node. A BPM node can have multiple instances of input and output links. Therefore, it can process multiple input requests concurrently. The number of BPM nodes in a BPM ring is subject to the actual requirements. BPM rings are policy-driven and dynamic. The BPM policy as mentioned in previous section is used to govern the information exchange and control signaling among BPM nodes. BPM rings can be used as a simple modeling vehicle of integrating BPM capabilities at various organizational levels, e.g., strategic, operational and execution.

BPM rings provide the means of building highly configurable and adaptive integration platform for BPM solutions. In our example, we have come up with 5 typical BPM service nodes in a BPM ring: (1) event processing service that takes raw data and produce qualified data to be further processed; (2) metric generation service that receives the qualified data and produced metrics; (3) situation detection service that analyzes incoming metrics and raise situations if needed; (4) action planning service that is triggered by situations and creates an action plan in order to resolve the situation; and (5) action rendering service that takes a group of actions from action planning service and actually renders them to the target business solutions. A BPM service node can process multiple input data requests based on the functionality to which it is aimed. Each service realizes grid specification and developed upon OGSA code base.

Implementation-wise, the BPM ring architecture is a physical star and a data processing ring. The BPM ring nodes are connected to a dispatching module called a Multi Node Access Unit (MNAU). Normally several MNAUs are connected in one BPM node while BPM links connect those MNAUs to the BPM nodes. This makes up the physical star. The control flow is rendered from one BPM node to the other through the MNAUs and each connected BPM links. The control flows of BPM ring realized by control tokens. Each BPM node on a BPM ring acts as both a data transformer and a repeater, receiving a series of data from one node and passing them on to the next. During this transformation/repeating process, if a ring node notices that it is the destination of the control flow (coded in the token), each data is copied into BPM data repository and the final data stream is altered slightly to let other ring nodes know that the control token was received. The control token is sent to each ring node in a specific order, known as the ring order. This ring order never changes unless another ring node joins or leaves the ring. Once the token reaches the last node in the ring, it is

sent back to the first node. This method of token passing allows each node to view the token and regenerate it along the way.

A BPM node is triggered when it receives a control token. This token gives the ring node permission to transform and transmit data. If there are more than one token residing within a BPM node. They will be queued up in local repository and will be processed in a first-come-first-serve fashion. However, some preemptive policies can be defined. One node on the network is the leader, and makes sure that the ring operates properly. This leader is called the BPM ring Leader. It performs several important functions including control token timing, making sure that control tokens and data don't circle the ring endlessly, and other maintenance duties. All nodes have the built-in capability to be the BPM ring Leader, and when there is no monitor on a ring, all the BPM nodes use special procedures to select one.

BPM Nets

Figure 2 illustrates a potential structure of BPM net formed by BPM rings and the interactions among them.

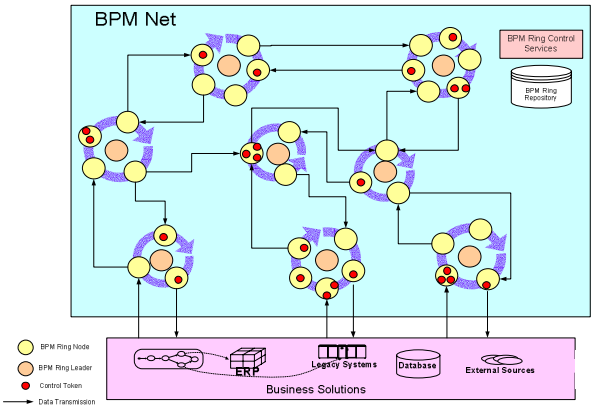

Fig. 2. BPM Net and BPM Rings.

Multiple BPM rings form a BPM net in that each BPM ring becomes a node and interactions among BPM rings constitute the links. While BPM rings capture the monitoring and control patterns of specific business situations (or exceptions), BPM net represents the pattern of communicating autonomous BPM rings in order to capture a global behavior of monitoring and control across business solution. Hence, a BPM net realizes the BPM capabilities for a business organization (enterprise). BPM rings collaborate with one another and aggregate into higher granularities. The structure of BPM nets can represent contractual bindings between business organizations (enterprises) and typically result in information exchange between business organizations (enterprises).

Formal BPM Net Model

A key goal of BPM net is to provide ubiquitous BPM services for target business solutions. Furthermore, the BPM net, is a dynamic and open environment where the availability and state of these services and resources are constantly changing. The primary focus of the BPM net model presented in this paper is to automatically create BPM policies (when possible) from the set of available services to satisfy dynamically defined monitoring and control objectives, policies and constraints. In the BPM net model, BPM services and policies can he dynamically defined. The pool of currently available BPM services is represented as a graph where the node represents services and the links, can be modeled as potential interactions.

To define BPM net, we need to define the relation, called *subsumption*, among BPM rings. For two messages M_1 and M_2, we define that M1 is subsumed by M2, (noted by $M_1 \sqsubseteq M_2$), if and only if for every argument *a* in the output message of M1, there is always an argument *b* in the input message of M2 such that either they have the same type or the type of a is the subtype of the type of *b*. Formally, $M_1 \sqsubseteq M_2 \Leftrightarrow \forall a \in M_1.\text{Output\_Arg}$

$$(\exists\, b \in M_2.\text{Input\_Arg s.t. } (type(a) = type(b)) \vee substype(a,b))$$

Similarly, for two services S1 and S2, we say that S1 is subsumed by S2 if for every message M1 in S1, there is a message M2 such that M1 is subsumed by M2. Formally, $S_1 \sqsubseteq S_2 \Leftrightarrow \forall M_1 \in S_1\ (\exists\, M_2 \in S_2 \text{ s.t. } M_1 \sqsubseteq M_2)$.

The formal definitions of BPM ring and BPM net are as follows:

1) A BPM ring $R_k = (S_k, C_k)$ where, S_k is a set of service nodes and C_k a set of service connection.

   a) Service set $S_k = \{s_{k,1}, s_{k,2}, \ldots, s_{k,n_k}\}$ where n_k is the number of functional stages in the ring R_k;

   b) Connection set $C_k = \{c_{k,1,2}, c_{k,2,3}, \ldots, c_{k,n_k-1,n_k}\}$ where $c_{k,i-1,i}$ connects $s_{k,i-1}$ and $s_{k,i}$. The data output of $s_{k,i-1}$ is the input of $c_{k,i-1,i}$ and the input of $s_{k,i}$ is the output of $c_{k,i-1,i}$.

2) A BPM net is a structure based on a service graph $N(B, \Sigma, \Phi)$ where B is the business solution that the BPM Net monitors and controls, Σ a set of BPM rings, and Φ a set of potential interactions among rings.

   a) The target business solutions B = {P, E} where P is set of probes that emit monitored data to BPM net and E a set of effectors that received control directives from the BPM net.

   b) The set of rings $\Sigma = \{R_i\}$ where each of R_i is associated with an order set of contextual data $\{Context(R_i)\}$.

c) The set of potential interactions among rings $\Phi = \{ L_{(i,x),(j,y)} \}$ such that R_i, $R_j \in R$ and x-th service of R_i connects to y-th service of R_j. Each connection is associated with a utility function to calculate the cost value $Cost(L_{(i,x),(j,y)})$.

3) In the net graph, $N(B, \Sigma, \Phi)$, the available services are nodes and interactions are edges. The edges $\{ L_{(i,x),(j,y)} \}$ are created at runtime when one of the following conditions hold

   a) Both $S_{(i,x)}$ and $S_{(j,y)}$ belong to the same ring, i.e., i = j and y = x+1.

   b) $S_{(i,x)}$ is subsumed by $S_{(j,y)}$, i.e., $S_{i,x} \sqsubseteq S_{j,y}$

4) The initial service S_0 of the ultimate BPM net is the service that can consume the output generate by the probes of the business solution P, hence, $S_0 \sqsubseteq P$.

5) The final service S_f of the of the ultimate BPM net is the service that produce the output to be consumed by the effectors of the business solution E, hence, $E \sqsubseteq S_f$

6) The chosen services from BPM net at run time form an execution path $\{S_0, S'_1, S'_2, \ldots, S_f\}$ in $N(B, \Sigma, \Phi)$

7) The costs of S_0 and S_f represent the costs of instrumentation of the target business solution. Assume the total cost of monitoring and controlling business solution *B* is constrained by a given value *CostBound* then we have the following relation for the final execution path:

$$Cost(S_{initial}) + Cost(S_{final}) + \sum_{i=1}^{n} Cost(S_i') \leq CostBound$$

The subsumption relationships among services can be used to generate candidate BPM services for the ultimate BPM net. The constraints among services are given by the users including the total execution cost of monitoring and controlling target business solutions. We single out the cost of the instrumentation of target business solution, which make it ready to be monitored and controlled by BPM net because of the high variability of such cost for different solutions. For the BPM net, the candidate execution paths can be generated from S_0 to S_f.

BPM Capabilities

The execution paths generated from BPM net based on constrains and goals defined in the BPM requirement actually manifest the *capabilities* of a BPM system on monitoring and controlling business solutions. As described in previous section, BPM policies are applied to multiple levels of emprise abstraction: strategy, operation, execution, and implementation. Each layer consists of corresponding BPM rings that are specialized in monitoring and controlling specific layer of enterprise resources.

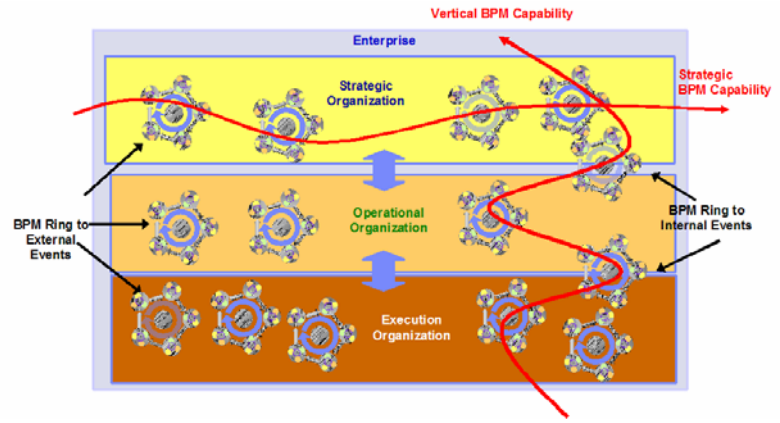

Fig. 3. BPM Capabilities.

Figure 3 illustrates the distribution of BPM rings in different enterprise layers. BPM capabilities can be defined either horizontally or vertically. Horizontal BPM capability is an execution path that consists of BPM rings exclusively of a specific layer, e.g. the strategic BPM capability. On the other hand, the vertical BPM capability is an execution path which contains the BPM rings across different layers. In the diagram, it is also indicated that some BPM rings are for processing external events and some for internal events among BPM rings.

Discussion

We have applied the concepts of BPM policies into real customer scenarios such as the one described in Section 1. A policy-driven BPM system makes it adaptive to monitor and control business solutions, which is particularly useful for the domain with high volatility of monitoring and control requirements. Crystallization of BPM policies into BPM rings and BPM net increases the modularity and reusability of BPM policies and consequently the system behavior. Formalization of BPM nets allows the dynamic formation of service execution and hence makes BPM system on demand monitoring and control system. The formal model of BPM nets also allows us to optimize the execution of BPM nets based on given constraints and cost bounds. Usually, the monitoring and control applications for specific business solution such as supply chain management systems are defined in an ad-hoc and static manner. A BPM solution is bound with a set of services at design time, which realizes the early binding of BPM policies with the underlying policy architecture. However, in an on-demand environment, the binding is not possible until the policies are discovered and enforced at run time. There are benefits and disadvantages on either approach. Early bindings facilitate the analysts to perform the policy impact at design time and hence imply an efficient implementation at run time. On the other hand, late bindings enable high flexibility of policy bindings with the policy architecture such as execution paths. Therefore, more adaptive BPM functionality can be enabled via policies.

4 Related Work

The policy-driven management model is recognized as an appropriate model for managing distributed systems [7][8]. This model has the advantages of enabling the automated management and facilitating the dynamic behaviors of a large scale distributed system. Policy works in standard bodies such as focus more on defining frameworks for traditional IT systems. Minsky and Ungureanu [9] described a mechanism called law-governed interaction (LGI), which is designed to satisfy three principles: (1) coordination policy needs to be coordinated; (2) the enforcement needs to be decentralized; and (3) coordination policies need to be formulated. LGI uses decentralized controllers co-located with agents. The framework provides a coordination and control mechanism for a heterogeneous distributed system. Verma et al. [10] proposes a policy service for resource allocation in the Grid environment. Due to the nature of Grid computing, virtualization has been greatly used for defining policy services in the paper. However, in contrast to their work, the BPM is aimed for providing policy framework for business activities instead of a service for system domain.

The Ponder Language [11] and Policy Framework for Management of Distributed Systems [12] address the implementation of managing network systems based on policies. Traditional grid based frameworks for enterprise [13] focus on distributed supercomputing, in which schedulers make decisions about where to perform computational tasks. Typically, schedulers are based on simple policies such as round-robin due to the lack of a feedback infrastructure reporting load conditions back to schedulers. However, the BPM system is governed by the BPM policies (BPM nets) that are a mode sophisticated policy than OGSA policy. ACE [14] presents a framework enabling dynamic and autonomic composition of grid services. AThe formal model of BPM nets has similar merits to their approach. However, our framework is aimed for composing monitoring and control systems for business solutions.

5 Conclusion

In this paper, we have described an approach of building an adaptive BPM policy architecture for managing business solutions. The system is designed, keeping in mind the need for multi-level of abstraction, various types of services, and different types of collaboration so that not only can BPM chores be quickly assembled and executed, but the configuration data can be deployed to the system dynamically. The dynamic interactions among services are captured in the BPM net in response to business situations that are detected from the set of observed or simulated metrics in the target business solutions. The BPM net model allows the composition of BPM services and resources using policies. We have defined a formal model for such purpose. Much more work remains to be done toward realizing complete and full implementation of BPM net. The future works include: automating the derivation of configuration model based on BPM policies, defining dynamic resource model and relations using ontological approach, applying model-

driven approach into the development of BPM applications, and developing BPM policy and configuration tools.

References

1 G. Lin, S. Buckley, M.Ettl, K. Wang. Intelligent Business Activity Management – Sense and Respond Value Net Optimization. To appear in: C. An, H. Fromm (eds.) Advances in Supply Chain Management. Kluwer (2004).

2 L.S.Y. Wu, J.R.M. Hosking, and J.M. Doll, "Business Planning Under Uncertainty: Will We Attain Our Goal?," IBM Research Report RC 16120, Sep. 24, 1990, Reissued with corrections Feb. 20, 2002.

3 "Business Process Execution Language for Web Services Version 1.1," http://www-106.ibm.com/developerworks/library/ws-bpel/

4 "Web Service Notification," http://www-106.ibm.com/developerworks/library/specification/ws-notification/ March, 2004.

5 "Open Grid Services Architecture," http://www.globus.org/ogsa/

6 H. Li, J.J. Jeng, "Managing Business Relationship in E-Services Using Business Commitments", Proceedings of Third International Workshop, TES 2002, Hong Kong, China, August 23-24, 2002, LNCS 2444, pages 107-117.

7 The IETF Policy Framework Working Group: Charter available at the URL http://www.ietf.org/html.charters/policy-charter.html

8 Distributed Management Task Force Policy Working Group, Charter available at URL http://www.dmtf.org/about/working/sla.php.

9 N.H. Minsky and V. Ungureanu, "Law-Governed Interaction: A Coordination and Control Mechanism for Heterogenous Distributed Systems," ACM Transaction on Software Engineering and Methodology, Vol. 9, No. 3, July, 2000, Pages 273-305.

10 N. Damianou, N. Dulay, E. Lupu, M. and Sloman, M., "The Ponder Policy Specification Language", *Proceedings of the Policy Workshop 2001*, HP Labs, Bristol, UK, Springer-Verlag, 29-31 January 2001, http://www.doc.ic.ac.uk/~mss/Papers/Ponder-Policy01V5.pdf

11 N. Damianou, "A Policy Framework for Management of Distributed Systems", PhD Thesis, Faculty of Engineering of the University of London, London, England, 2002, http://www-dse.doc.ic.ac.uk/Research/policies/ponder/thesis-ncd.pdf

12 D. Verma, "A Policy Service for Grid Computing," M. Parashar (Ed.): GRID 2002, LNCS 2536, pp. 243–255, 2002.

13 Helal, S. et al, The Internet Enterprise, In Proceedings of the 2002 Symposium on Application and the Internet (SAINT2002).

14 R. Medeiros, et. al "Autonomic Service Adaptation in ICENI Using Ontological Annotation," in the Proceedings of the Fourth International Workshop on Grid Computing (GRID 2003), pages 10-17, Phoenix, Arizona, November 17, 2004.