

Distributed Policy Specification and Enforcement in Service-Oriented Business Systems

W. T. Tsai, Xinxin Liu, Yinong Chen

Computer Science & Engineering Department

Arizona State University, Tempe, AZ 85287-8809, USA

Abstract

Service-Oriented Computing (SOC) and Web Services (WS) provide a flexible computing platform for electronic business and commerce. Introducing policy-based computing to service-oriented business systems adds another dimension of flexibility and security. While service composition and re-composition in service-oriented business systems allow major system reconstruction, policy-based computing can better deal with the small and routine changes of business processing. This paper reports our latest research on integrating policy-based computing into service-oriented business system and discusses its feasibility, benefits and cost. Under this research, we designed a policy specification and enforcement language PSEL for specifying system constraints and business rules. We implemented a runtime environment in which a service-oriented business system can be modeled, analyzed, deployed, and executed with policy enforcement. Automated tools have been developed to facilitate the entire development process. The cost of policy-based computing is experimentally evaluated.

Keywords: Service-oriented architecture, Web services, policy-based computing, policy specification language, distributed policy enforcement

1. Introduction

Service-Oriented Computing (SOC) and Web Services (WS) have become the powerhouse behind the electronic business and commerce in recent years. The major enabling technique is the dynamic service composition and re-composition that perfectly match the dynamic natures of electronic business and commerce. Under SOC and WS architecture, a new service can be composed using existing services, an existing service can be removed, modified, or re-composed at runtime and in real-time. Although dynamic re-composition can provide the crucial flexibility, it is not efficient to perform re-composition frequently for small and routine changes.

In electronic business and commerce, there are many rules, regulations, and policies that govern the business practice and transactions. For example, 90-day return policy, 30-day payment on delivery of goods, pay-before-shipping, credit overdue charge and rate, credit ranking requirement, password composition, and so on. Obviously, many of these rules, regulations, and policies can change and be enabled or disabled from time to time. If they are hard-coded into the services, it will be difficult to add, remove, change, enable and disable these policies [11].

To tackle these problems, policy-based computing has been increasingly used in computing systems (such as operating systems, distributed systems, and computer networks) for dynamically specifying and enforcing system constraints. A *policy* is a statement of the intent of the system user or administrator, specifying how she or he wants to use the system [1]. A policy-based system allows the system itself and policies be specified separately, and policies are enforced dynamically during the system execution. A number of Policy Specification Languages (PSL) [2][3][6] have been proposed to be used in policy-based systems. Most policy-based systems are centralized, which means a centralized policy manager maintains a policy database and enforces policies during the system execution. The advantage of the centralized policy systems is its simplicity in management. However, centralized policy systems will not work for a service-oriented system for the following reasons:

- A service-oriented system is normally a large distributed system with a large number of policies defined to ensure the synchronization, consistency, and security of the distributed system. As more policies are added into the centralized policy database, the policy-related computing will be extremely time-consuming and the centralized policy management becomes the bottleneck of the entire system.
- A large service-oriented system may contain hundreds of components connected through LAN

or Internet. Centralized policy management will cause many unnecessary communications.

Ponder [3][4][5] and Rei [6][7][8][9][10] are two widely used policy languages that can be applied to distributed systems. Policies specified in Ponder or Rei are deployed to policy engines that can be queried by distributed components. However, distributed components can freely bypass policy engines, leaving policies un-enforced. In our framework, policy engines are seamlessly integrated to distributed agents on which distributed components are deployed, which enables policy engines to monitor every action performed by a distributed component. Furthermore, Ponder and Rei provide no systematic policy engineering. There are no easy ways to conveniently integrate policy specification and enforcement to distributed systems. Policies and distributed systems do not share a system model, which forces policy engines to do extra work to understand the semantics of distributed systems. Our framework provides a shared system model for both distributed systems and policies specified in PSEL (Policy Specification and Enforcement Language). In addition,

a complete process of policy engineering enables policies to be integrated into distributed systems with little effort. Lastly, both Ponder and Rei address few core issues of distributed policies, including policy dispatch, synchronization and enforcement.

This paper introduces a distributed policy-based framework with hierarchical service-oriented architecture to address the problems mentioned above. Figure 1 illustrates the overall policy framework. In this framework, services are at different levels. A higher level service can invoke a lower level service. At each level, a service (deployed to a distributed agent) interacts and collaborates with other services to complete designated tasks. There is a local policy engine associated with each distributed agent, which ensures that policies specified for the local services are enforced. At each level, there is also a global policy that enforces policies involve at least two services. During system execution, the policy service along with local policy engines act as the policy kernel that ensures every policy in the system is enforced and no actions can bypass the policy kernel.

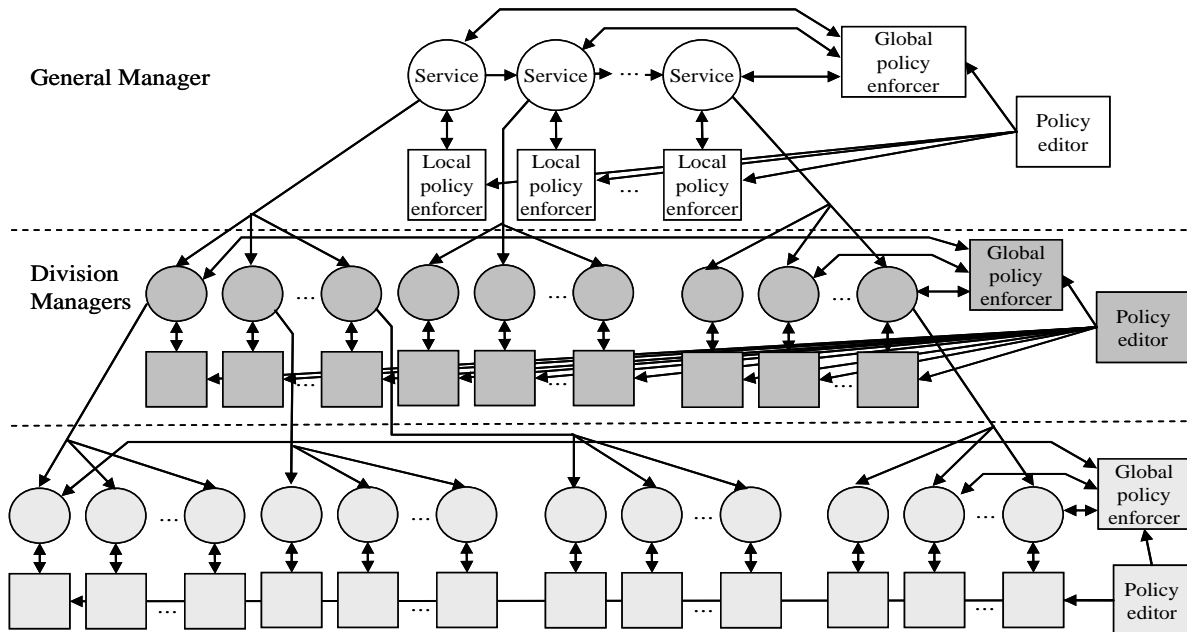


Figure 1. Policy-Based Framework for Service-Oriented Systems

To facilitate the application of the policy-based framework in service-oriented systems, we designed and implemented a set of supporting tools and services:

- ACDATE (Actors, Conditions, Data, Actions, Timing, and Events) scenario language for modeling and specifying service-oriented computing systems. The architecture is modeled by

individual ACDATE elements and the behaviors are modeled by system scenarios.

- PSEL language is used to specify system constraints and business rules. PSEL shares the same syntax and semantics with ACDATE.
- REI (Runtime Execution Infrastructure) provides services that support the execution of service-

oriented systems with distributed policy enforcement.

- Code Generation Service generates executables from the specification in ACDATE language.
- Deployment Service deploys the executables to distributed agents for execution.
- Event Service is in charge of the communication among distributed agents.
- The Information Service is responsible for providing global information to distributed agents and helping locate services (like a UDDI server).
- The Policy Service manages policies, dispatches and synchronizes local policies, and it is also in charge of enforcing global policies.

The framework is an integrated framework for supporting the specification of service-oriented systems and policies, automated code generation, deployment, execution, and policy enforcement. Once a service-oriented system is specified by ACDATE modeling and specification language, the rest of work is automatically taken care of by the tools in the framework. Different from the traditional “model-code-run” approach, our framework removes the manual “code” step with the Code Generation Service that automatically generates executables at the service composition level. These automated tools significantly reduce the system development overhead. It is assumed that the atomic services have been developed. They are either wrapped code from legacy software components or newly developed basic services.

Furthermore, our framework includes a set of tools performing Verification and Validation (V&V) [12] that ensures the completeness and consistency of the modeling and the compliance with the specification. Verification is done by model checking and the completeness and consistency analysis. Validation is done by testing during the execution. Test cases used in testing is generated during verification.

Section 2 presents a system with the Service-Oriented Architecture (SOA). Section 3 introduces the process of ACDATE modeling and policy specification in PSEL. Section 4 presents the Runtime Execution Infrastructure. Section 5 discusses the Policy Service. The policy enforcement is elaborated in Section 6. Section 7 presents experiment data and analyses.

2. An Example of SOA System

As an example of SOA system, this section presents the Internet Parts Ordering (IPO) system [18]. This

example will be used throughout this paper.

The IPO standard is developed by Automotive Aftermarket Industry Association (AAIA) to promote electronic commerce between participants in the automotive aftermarket [19]. The IPO implements SOA based on WS, which facilitates the e-business among participants (including parts manufacturers, warehouse distributors, and retailers). A typical scenario in the IPO is described as follows:

When the system at a retailer (retailer system for short) detects the number of an auto part in stock goes below the threshold, it sends a RequestForQuote message to a distributor’s WS (distributor system for short) for parts information and receives a quote. The retailer system can then send a PurchaseOrder message to place an order. On receiving the purchase order, the distributor system sends an acknowledgement as well as the invoice back to the retailer system. The distributor system checks the stock of the requested auto part and informs the shipping department if the purchase order can be filled. Otherwise, the distributor system will send a PartOrder message to the manufacturer’s WS for more parts. The distributor’s shipping department processes the order and ships auto parts if the retailer has paid the invoice. Once getting shipped, the distributor system will send the retailer system an order shipment notification.

Other typical scenarios in the IPO could be: a retailer queries the order status, a retailer changes or cancels the purchase order, etc. After analyzing the IPO, several policies are extracted and listed in Table 1.

Table 1. IPO Policies

Policy01	On receiving a purchase order, the distributor system must send an acknowledgement
Policy02	If the stock of an auto part is not below the threshold, the retailer system must not send a purchase order
Policy03	The purchase price must be less than the retailer’s account balance
Policy04	Action PayInvoice must be performed before Action ShipAutoPars
Policy05	The purchase order can only be sent once
Policy06	The retailer can access the quote information only during business hours

3. SOA System Modeling

In our policy-based framework, an SOA system is modeled and specified using ACDATE modeling and specification language. The system structure is modeled by ACDATE elements. System behaviors and system constraints are modeled through scenarios and policies,

respectively. The three steps of the ACDATE modeling process are illustrated by Figure 2. After the system requirements are analyzed, the structure of an SOA system is decomposed to ACDATE elements. Meanwhile, system behaviors (scenarios) and system constraints (policies) are also extracted. The formal specification of scenarios and policies are then reconstructed based on ACDATE elements. The detailed process is discussed in [10].

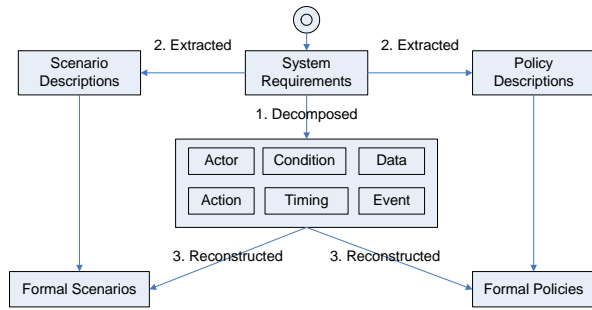


Figure 2. ACDATE Modeling

Policies are constraints that specify 1) actors' responsibilities (obligation policies), 2) actors' rights (authorization policies), or 3) constraints that have to be enforced (system constraints). In Table 1, Policy01 and Policy02 are obligation policies. Policy03, Policy04 and Policy05 are system constraints. Policy06 is an authorization policy. These policies are formally specified in PSEL, which is also discussed in [10].

4. Runtime Execution Infrastructure

The Runtime Execution Infrastructure (REI) provides a platform for multi-agent, distributed, interactive, event-driven execution of SOA systems modeled by ACDATE language. As showed in Figure 3, REI consists of 1) a series of runtime services and 2) a group of distributed agents. After an SOA system is modeled by ACDATE, the system model will be sent as inputs to REI. REI provides runtime services that support the execution of an SOA system with distributed policy enforcement.

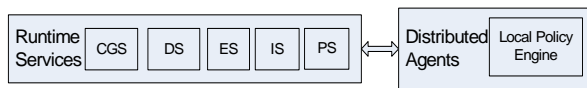


Figure 3. Runtime Execution Infrastructure

4.1 Runtime Services

Code Generation Service (CGS) generates executables for an SOA system modeled by ACDATE. Each ACDATE element and scenario is translated into a

service that implements public interfaces, which can be interpreted and executed by runtime services and distributed agents. Since policies are to be dynamically loaded, interpreted, and enforced, CGS does not generate executables for policies.

Deployment Service (DS) deploys the executables of an SOA system to distributed agents. In the ACDATE model, conditions, data, actions, timings, events, and scenarios have an actor as their owners. DS packages an actor along with its elements (including conditions, data, actions, timings, events and scenarios) and deploys the package to distributed agents according to a deployment configuration file. A distributed agent can host one or more actors.

Events are the only means of communication among distributed agents. Events are triggered (e.g., a service invocation) while scenarios are executed by distributed agents and sent to the *Event Service (ES)* for forwarding. On receiving a triggered event, ES determines the targeted distributed agents and forwards the event to them. A distributed agent must register events it is interested in with ES, which gives ES the knowledge where to forward a given event. Figure 4 illustrates an example of event passing through ES.

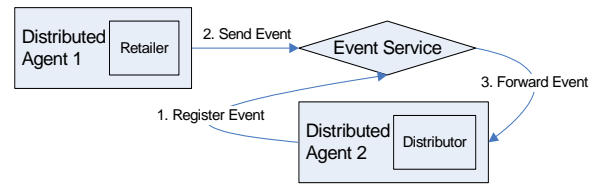


Figure 4. Event Service

Information Service (IS) provides the global information to all distributed agents. During the system execution, a distributed agent may need to access data deployed to other distributed agents. Such access requests are directed to IS that is in charge of assessing the requests and returning the data. Since IS maintains the global data, any data changes made by distributed agents are submitted to IS. Another important service provided by IS is the service location. When a distributed agent needs to access other services, it may need to send a request for service location to IS that returns the matched services.

Policy Service (PS) takes care of policy management, local policy dispatch, synchronization, and global policy enforcement. When policies are changed (added, deleted, or updated), PS is responsible for dispatching the changed local policies to the corresponding local policy engines. Meantime, PS provides a mechanism to synchronize local policies received by different local policy engines. During the system execution, PS is the

point where global policies are enforced (local policies are enforced by the local policy engine at the distributed agents). Policy Service is discussed in detailed in Section 6.

4.2 Overall Process

1. An SOA system is modeled and specified through ACDATE elements and scenarios. System constraints are specified by policies in PSEL.
2. Code Generation Service translates the ACDATE elements and scenarios into executables that can be interpreted and executed by distributed agents.
3. The executables are packaged and deployed by the Deployment Service to distributed agents according to the deployment configuration file.
4. Local policies are dispatched by the Policy Service to local policy engines for enforcement. Global policies are maintained by the Policy Service and enforced by the global police engine.
5. The distributed agents register events with the Event Service, then interpret and execute system scenarios with local policy engines enforcing local policies. Global policies are enforced by the Policy Service. Communication among distributed agents is through event triggering and event forwarding provided by the Event Service. The Information Service provides global data if requested by distributed agents, as shown in Figure 5.

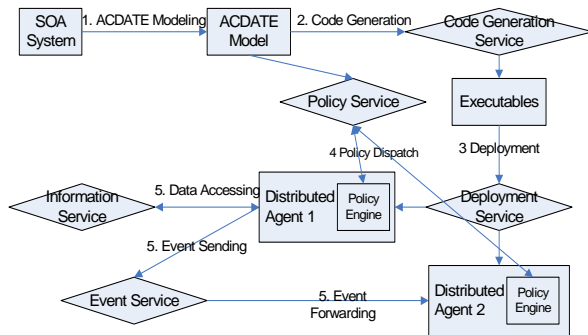


Figure 5. Runtime Services

5. Policy Service

The responsibilities of the Policy Service are to 1) manage policies, 2) dispatch local policies to local policy engines, 3) synchronize local policies, and 4) enforce global policies.

5.1 Policy Dispatch

As shown in Figure 6, there is a dispatch component

in the Policy Service responsible for dispatching policies to policy engines. Each distributed agent is associated with a local policy engine where local policies are dispatched and enforced. The Policy Service itself also contains a global policy engine to enforce global policies. The dispatch component determines whether a policy is global or local.

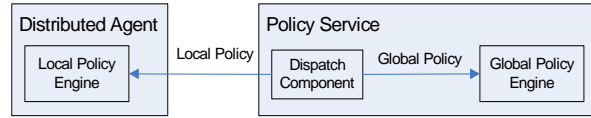


Figure 6. Policy Dispatch

Global policies are those that involve two or more distributed agents and can not be enforced by one local policy engine alone. For instance, Policy04 in table 1 is a global policy because the actors (owners) of the two actions are different (the actor of “PayInvoice” is a retailer while the actor of “ShipAutoParts” is a distributor). Since these two actions are packaged into different actors and deployed to two distributed agents, each distributed agent does not have knowledge of the other. If this policy had been dispatched as a local policy, neither distributed agent would know how to enforce it. Only the Policy Service, as the coordinator of all local policy engines, has the overall picture of the system. Therefore, global policies will not be dispatched to distributed agents.

Local policies are those applied to an individual actor and can be enforced by a local policy engine. For instance, Policy01 in table 1 is a local policy because the actor (owner) of “SendAcknowledgement” and the action itself are both deployed to the same distributed agent that knows how to enforce this policy without any knowledge from other distributed agents. The Policy Service identifies all distributed agents with an actor that takes the “retailer” role, and dispatches policy01 to policy engines at these distributed agents. During the system execution, the policy engines with policy01 enforce it on receiving event “ReceivePurchaseOrder”.

5.2 Policy Dispatch Strategy

The type of a policy determines whether it is a global or local policy. Obligation policies (OP) specify a role’s responsibilities and they are dispatched by the Policy Service as a local policy to all distributed agents that have an actor who takes this role.

For authorization policies (AP) specified in RBAC [14], they specify a role’s rights to perform certain actions. They are also dispatched as a local policy to all distributed agents that have an actor who takes this role. Authorization policies specified in BLP [13] specify

security levels to both actors and data. Since actors are allowed to access data belonged to other actors that are deployed to different distributed agents, authorization policies specified in BLP can not be enforced by the local policy engine in a distributed agent. They must be global policies and enforced by the Policy Service. Similarly, conditional authorization policies (CAP) have to be enforced as global policies because they allow / deny actors to access data located at different distributed agents.

System constraints (SC) on data specify the legitimate range that data must conform to. Since each datum has an actor, a system constraint on data can be dispatched as a local policy to where its owner is deployed. System constraints on actions specify the temporal constraints on actions. If these actions are deployed to the same distributed agent, it is dispatched as a local policy. Otherwise, it is a global policy. Table 2 summarizes the policy dispatch strategy.

Table2. Global and Local Policies

Policy Type	Global Policy (Policy04, 06)	Local Policy (Policy01, 02, 03, 05)
OP		X
AP in BLP		X
AP in RBAC	X	
CAP	X	
SC on Data		X
SC on Actions	X	X

5.3 Passive Policy Object and Active Policy Service

Policies in an SOA system can be treated as either passive objects or active services [17]. As passive objects, policies are specified and passed around as data such as XML text. In contrast, policies treated as active services can be invoked by distributed agents like other services to enforce system constraints. Our framework supports both approaches. Section 7 lists the experiment data collected with these two approaches.

Policy as Passive Objects: During the initialization phase, the Policy Service dispatches local policies as XML text to the local policy engine at a distributed agent. On receiving policies, the policy engine creates a local policy database. During the system execution, the Policy Service will inform local policy engines if there is any policy that has been changed. However, policy engines located at different distributed agents need to synchronize local policies to ensure policy consistency. Standard synchronization protocols (such as 2PC) can be employed to synchronize local policies. The benefit

brought by treating policies as passive objects is the lower time complexity and communication cost because local policies are enforced locally without having to connect to the Policy Service.

Policy as Active Services: The Policy Service will not dispatch local policies to local policy engines at the initialization phase. Instead, policies are requested as a service from the Policy Service every time before a scenario is executed. The remarkable benefit brought by this approach is that synchronization is not needed because local policy engines do not maintain a local policy database that could be inconsistent with those at different policy engines. Furthermore, policy changes can easily take effect with new services replacing old services. However, this approach increases the overhead (including system time and communication cost).

6 Distributed Policy Enforcement

During the execution, local policies and global policies are enforced by policy engines at distributed agents and the Policy Service, respectively, as shown in Figure 7. No matter where the policy engine is located, policies are enforced in the same way. Therefore, subsequent discussions will not distinguish the local policies and global policies.

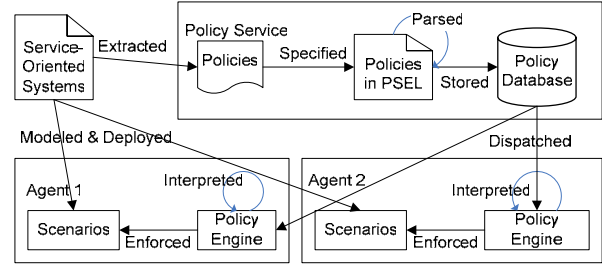


Figure 7. Policy Enforcement Framework

Policy Parsing and Storage: After policies are extracted and specified in PSEL, the Policy Service parses policies for correctness, completeness, and consistency. On successful parsing, policies are translated into XML and stored into the policy database. The policy parser is implemented by ANTLR (ANOther Tool for Language Recognition). According to the policy syntax, ANTLR creates an Abstract Syntax Tree for each policy. Policy elements (roles, actions, conditions, etc) are extracted by traversing the tree.

Policy Registration & Policy Map: On receiving policies from the Policy Service, policies are registered with the policy engine, which lets them know when to trigger policy enforcement. Three types of ACDATE elements can trigger the policy enforcement: *events*,

actions, and data. Events trigger positive obligation policies; actions trigger negative obligation policies, authorization policies, and system constraints on actions; data changes trigger system constraints on data. To improve performance, a policy map is created, mapping an event, action or datum to a list of relevant policies to which it has been registered with. All policies registered to a particular event, action or datum form a linked list. The linked list and the ID of the action, event or datum are then organized as a policy map. When policy enforcement is triggered, the policy engine maps the ID of the event, action or datum to the linked list, and enforces all policies in the linked list.

Policy Enforcement: During the execution, distributed agents keep monitoring and trigger the local policy enforcement by invoking the EnforcePolicy() method in the policy engine when an event is triggered, an action is performed or a datum is changed. The EnforcePolicy() method uses the ID of the event, action or datum to look up its policy map and maps the ID to a list of relevant policies that are then enforced one by one. Policy violations are recorded in a log and returned to distributed agents. After local policies are enforced, distributed agents invoke EnforcePolicy() method in the Policy Service for global policy enforcement.

7 Experiment Data & Analyses

Policy-based computing brings dependability and flexibility to SOA systems by allowing system constraints to be dynamically specified and enforced. However, dependability and flexibility comes with a cost: the increased system overhead. The system overhead comes in two forms: system running time and communication cost. We implemented the IPO and conducted a series of experiments in 3 configurations to explore the system overhead [20]:

- Config1: No policy enforced
- Config2: Policy enforced as passive objects
- Config3: Policy enforced as active services

For each configuration, we simulated the execution

of IPO and collected system running time as well as the communication cost (data presented in Table 3). The system running time is the time elapsed between the first event is triggered and the last scenario is executed. The policy time is part of the system running time spent on policy-related computing. The communication cost only includes the data packages transferred as part of policy enforcement (such as policy dispatch).

System Running Time: in Config1, the simulation takes the shortest time because there is no overhead brought by policy-based computing. In Config2 and Config3, the system running time is increased by 11% and 15%, respectively, compared to Config1 due to policy-related computing. Config3 takes more time than Config2 because local policy engines need to connect to the Policy Service for updated policies every time before a scenario is executed. However, in a large-scale SOA system where policy changes take place frequently, Config3 can significantly reduce the system running time since no policy synchronization is required.

Policy Time: is deduced by subtracting the system running time without policy enforcement from the system running time with policy enforcement, which represents the time spent on policy-related computing (including policy dispatch, policy enforcement, policy synchronization, miscellaneous). In Config3, the policy time is 39.5% (162ms versus 226ms) more than that of Config2 mainly due to the communication time incurred by the policy dispatch. Table 3 also lists the percentage breakdown of the policy time to its 4 categories of policy-related computing.

Communication Cost: in Config1, there is no communication cost involving policy enforcement. In Config2, communication cost mainly comes at the initialization phase when the Policy Service dispatches local policies to local policy engines. The remaining communication cost comes with the local policy synchronization when policies get changed. In Config3, communication cost mainly comes when local policy engines request for policies from the Policy Service.

Table 3. Experiment Data

Configurations		No Policy (Config1)	Passive Object (Config2)	Active Service (Config3)
System Running Time (ms)		1478 (100%)	1640 (111%)	1704 (115%)
Policy Time (ms)	Total Policy Time	0	162	226
	Dispatch	0	28 (17.3%)	84 (37.2%)
	Enforcement	0	83 (51.2%)	122 (54.0%)
	Synchronization	0	33 (20.4%)	0 (0.0%)
	Miscellaneous	0	18 (11.1%)	20 (8.8%)
Communication Cost (KB)		0	6.86	18.29

8 Summary

This paper presents a framework that facilitates the development of policy-based service-oriented business systems. The system services are specified by the ACDATE elements and scenarios that are translated into the executables. PSEL is designed for specifying policies. REI provides a series of services that support system execution with distributed policy enforcement. The Policy Service and policy engines act as the policy kernel that no actions can bypass. Such a framework is useful for B2B commercial systems because it supports dependable, flexible, policy-based, service-oriented computing. The paper also explores the System overhead of the policy-based computing.

References

- [1] C. Pleeger, *Security in Computing*, 3rd Edition, Prentice Hall PTR, 2003.
- [2] N. Damianou, A. Bandara, M. Sloman and E. Lupu, "A Survey of Policy Specification Approaches", Technical Report, Department of Computing at Imperial College of Science Technology and Medicine, 2002.
- [3] N. Damianou, N. Dulay, E. Lupu and M. Sloman, "The Ponder Policy Specification Language", Proceedings of Workshop on Policies for Distributed Systems and Networks, 2001, pp. 18-38.
- [4] L. Lymberopoulos, E. Lupu and M. Sloman, "Ponder Policy Implementation and Validation in a CIM and Differentiated Services Framework", Proceedings of IFIP / IEEE Network Operations and Management Symposium, April 2004, pp. 31-44.
- [5] L. Lymberopoulos, E. Lupu and M. Sloman, "an Adaptive Policy Based Framework for Network Services Management", Plenum Press Journal of Network and Systems Management, Special Issue on Policy Based Management, Vol. 11, No. 3, September 2003, pp. 277-304.
- [6] L. Kagal, "Rei: A Policy Language for the Me-Centric Project", Technical Report, HP Laboratories, 2002.
- [7] L. Kagal, T. Finin, and A. Joshi, "Declarative Policies for Describing Web Service Capabilities and Constraints", Proceedings of W3C Workshop on Constraints and Capabilities for Web Services, October 2004.
- [8] L. Kagal, T. Finin and A. Joshi, "a Policy Based Approach to Security for the Semantic Web", Proceedings of the 2nd International Semantic Web Conference, September, 2003, pp. 402-418.
- [9] A. Patwardhan, V. Korolev, L. Kagal and A. Joshi, "Enforcing Policies in Pervasive Environments", Proceedings of International Conference on Mobile and Ubiquitous Systems: Networking and Services, August 2004, pp. 299-309.
- [10] L. Kagal and T. Finin, "Modeling Conversation Policies using Permissions and Obligations", Proceedings of AAMAS 2004 Workshop on Agent Communication, July 2004.
- [11] W.T. Tsai, X. Liu, Y. Chen, R. Paul, "Simulation Verification and Validation by Dynamic Policy Enforcement", Proceedings of the 38th Annual Simulation Symposium, April 2005, pp. 91-98.
- [12] W. T. Tsai, X. Wei, Y. Chen, B. Xiao, R. Paul, and H. Huang, "Developing and Assuring Trustworthy Web Services", Proceedings of the 7th International Symposium on Autonomous Decentralized Systems, April 2005, pp. 43-50.
- [13] D. Bell and L LaPadula, "Secure Computer System: Unified Exposition and Multics Interpretation", Technical Report, MITRE Corporation, March 1976.
- [14] R. Sandhu, E. Coyne, H. Feinstein and C. Youman, "Role-Based Access Control Models", IEEE Computer, Volume 29, Issue 2, February 1996, pp. 38-47.
- [15] B. Logan and G. Theodoropoulos, "the Distributed Simulation of Multi-Agent Systems", Proceedings of the IEEE, Volume 89, Issue 2, February 2001, pp. 174-185.
- [16] W.T. Tsai, L. Yu, A. Saimi and R. Paul, "Scenario-Based Object-Oriented Test Frameworks for Testing Distributed Systems", Proceedings of IEEE Future Trends of Distributed Computing Systems, May 2003, pp. 288-294.
- [17] W.T. Tsai, R. Paul and J. Bayne, "The Impact of SOA Policy-Based C2 Interoperation and Computing", Proceedings of the 10th International Command and Control Research and Technology Symposium (ICCRTS), June 2005.
- [18] Automotive Aftermarket Industry Association, <http://www.aftermarket.org/>
- [19] After Market Business, <http://aftermarketbusiness.com/>
- [20] Software Research Lab at Arizona State University, <http://asusrl.eas.asu.edu/PSEL/>