

# Object by Value Transfer Mechanisms for Obligation Policy Enforcement Object Loading

Mirko Randic, Marijan Kunstic, Bruno Blaskovic

University of Zagreb  
Faculty of Electrical Engineering and Computing  
Unska 3, HR-10000 Zagreb, Croatia  
E-mail: {mirko.randic | marijan.kunstic | bruno.blaskovic}@fer.hr

**Abstract**— Adequate architecture and technology can significantly improve adaptability and performance of the policy-based management systems. In this paper we present how policy enforcement point operation can be improved by applying the "object by value" transfer mechanisms. Policy enforcement object transferred and loaded into the policy enforcement point as a stateful object can support adaptable behaviour of the point. We have experimented with two standard object by value technologies: CORBA valuetypes and Java serialization on RMI. Examples in this paper refer to CORBA valuetypes only.

**Index Terms**— policy-based management, obligation policy, enforcement agent, object by value, CORBA valuetypes

## I. INTRODUCTION

Policy-based management is one of the latest developments in networks and distributed software systems management. It seems to be a very promising solution for managing large-scale distributed systems. Such management systems are self-adapting and can dynamically change their behaviour. The focal point in the area of policy-based management is the notion of *policy* as a means of driving management procedure. Furthermore, issues of an architecture and technology for policy-based management system building are very important for overall system performance.

In this paper we present an approach to operation improvement of the policy enforcement points. Policy enforcement point represents an important building block in standard architecture of the policy-based management system. We show how the operation can be improved by applying "object by value" transfer mechanisms for loading objects representing obligation policies into the policy enforcement points. Transferred as a stateful policy enforcement objects, they can support adaptable behaviour of the enforcement points [6].

In section 2 we present principles and standard architecture related to policy-based management systems.

We describe context of the policy enforcement points and their role in the system. As an obligation policy specification language we use, without any modifications, the language named PONDER developed at the Imperial College of Science Technology and Medicine, London [1]. So, we present it in short. In the section 3 we propose architecture for the policy-based management system. This architecture promotes object by value implementation of the policy enforcement objects. We have experimented with two standard object by value technologies: CORBA valuetypes and Java serialization on RMI. Examples in this paper refer to CORBA valuetypes only. Section 4 is reserved for conclusion with a comment on future work directions.

## II. PRINCIPLES AND STANDARD ARCHITECTURE

Generally, management system architecture specifies three kinds of entities: managers (entities which make management decisions), classical agents (entities without decision abilities; operation performers or notification senders only) and managed objects (represent managed hardware or software entities). A standard management control loop reflects the activity of managers (Fig. 1).

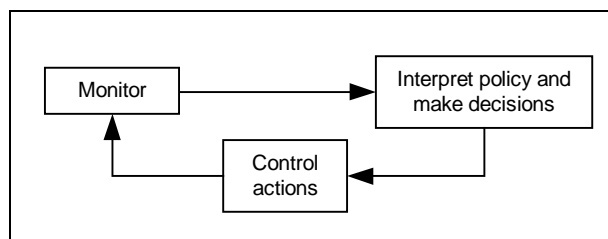


Figure 1: Management control loop.

Managed system monitoring includes current state and events monitoring. It is essential for all aspects of management [5], [2]. Events in form of notification message, time expiration, etc. stimulate managers to interpret policies that influence their decisions to perform management actions. Separating policies from managers that interpret them permits modifying policies to change

the behaviour and strategy of the management system without recoding the managers.

Figure 2 shows the policy-based management architecture defined within the IETF framework [8], which we treated as relevant in our work. The architecture specifies four key functional blocks: policy management application (PMA), policy repository (PR), policy decision point (PDP) and policy enforcement point (PEP). The PMA is expected to provide a graphical user interface to allow administrator to specify the policies (1), translate the input into a policy repository schema (usually LDAP, Lightweight Directory Access Protocol) and store them in the policy repository (2). However, the PMA can also be used to determine the association between the policies and the different devices (targets) to which the policies are applicable and to inform the relevant policy consumers – PDPs (3). The PR is a storage that is used for policy retrieval performed by the PDPs (4). It is assumed that policies are objects stored in an LDAP directory service which guarantee fast and efficient content retrieval. PDP also referred as a policy consumer, retrieves policies from the policy repository, interprets the policies and send decisions to PEP to enforce them. A PDP may need to translate policy rules received from the repository to a format that is understood by the corresponding PEP's. Furthermore, a PDP performs the function of receiving policy decision request from PEPs (5) and returning policy decision to them (6). PDPs also send asynchronous policy decisions based on updates or external requests (e.g. from external managers). Policy enforcement is realized by PEPs applying actions according to the PDP's decisions.

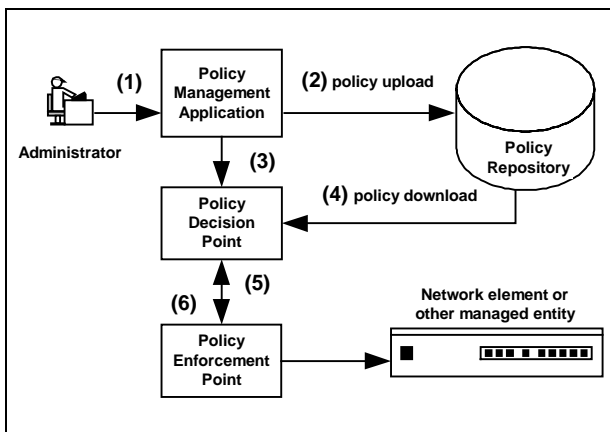


Figure 2: IETF policy framework.

Comparing with the classical management architectures, PDP can be considered as a mid-layer manager, while *PEPs are agents responsible for a set of managed objects*.

IETF framework suggests no implementation details such as distribution, platform, protocols or language. In our CORBA-based implementation we use the notion of

triggers that can be CORBA events, time expiration, explicit managed system's requests or external requests.

Generally, architectures for policy enforcement are moving towards strongly distributed paradigms, using technology such as mobile code, distributed objects, intelligent agents or programmable networks.

#### A. Basic of the PONDER Policy Specification Language

In large-scale systems it is not practical to specify policies for individual objects and so there is a need to be able to group objects to which a policy applies. *Domains* provide a means of grouping objects to which policies apply and can be used to partition the objects in a large system according to geographical boundaries, object type, responsibility and authority or for the convenience of human managers [7]. In Ponder, path names are used to identify domains. For the domains specification we use domain scope expression as defined in [1].

Obligation policies are event-triggered and specify the actions that must be performed by automated manager components (Policy Manager Agents) in subject domains on objects in the target domains. Events can be simple, i.e. an internal timer event, or an external event notified by monitoring (event) service components. The subject of a policy specifies the policy manager agents to which the policy must be distributed for interpretation. The target specifies the object on which the policy actions are to be performed. Both subject and target can be defined using a domain scope expression that identifies a set of objects in terms of union, difference and intersection operators over sets of domains and objects [1]. Actions can be specified by concurrency operators (sequential or parallel execution). The Ponder obligation policy specification syntax is shown on Figure 3:

```
inst oblig policyName {
  on event-specification;
  subject [<type>]domain-scope-expres.;
  [target [<type>]domain-scope-expres.;]
  do obligation-action-list;
  [catch exception-specification;]
  [when constraint-Expression;]
}
```

Figure 3: The Ponder obligation policy specification

Constraints are optional and can be specified to limit applicability of policies based on time or values of the attributes of the objects to which the policy refers. A subset of the Object Constraint Language (OCL) [3] is used to specify constraints in Ponder.

### III. SYSTEM ARCHITECTURE

In this section we describe architecture for building management system based on obligation policies. Obligation policies are applicable in the areas of configuration management, network management and distributed application management and follows an event-condition-action paradigm. They specify what activities a subject (manager) must or must not do to a set of target objects. Other important category, so called authorisation policies (specify what activities a subject is permitted or forbidden to perform on a set of target objects) can be supported by the same architecture by adequate extensions. Figure 4 shows the proposed obligation policy-based management system architecture.

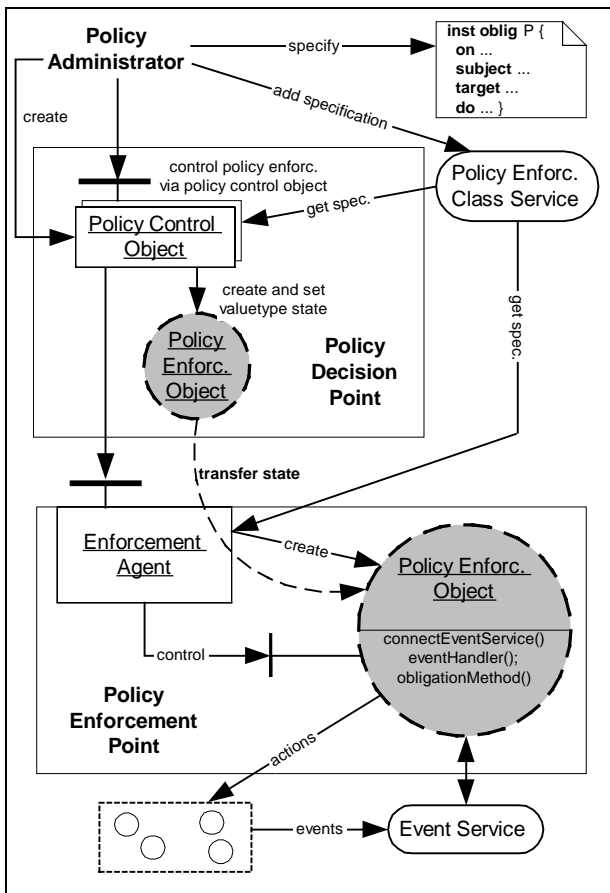


Figure 4: Policy-based management system architecture

Policy administrator specifies policies and creates corresponding policy control objects. Figure 5 shows two instances of the obligation policy specification. One represents time-triggered policy specification and the other policy specification triggered by the external event.

```
inst oblig HigherTariffPolicy {
  on Timer.at("07:00");/*at7 every morn.*/
  subject s=/AllRegions/TariffAgents;
  target
  t=/AllRegions/UserProxyDev/chargeSect;
  do t.setHigherTariff();
}
```

```
inst oblig OverloadHandler {
  on overload_event(Event e);
  subject s=ServerAgent;
  target t=ServerSystem;
  do t.startOneRequestOneServantMode()->
  s.log(e);
  when Time.between("07:00", "18:00");
}
```

Figure 5: Obligation policy examples

As we stated before, the concurrency operators can specify actions. In this example a sequential execution operator is used (**do** clause in the **OverloadHandler** policy).  $a_1 \rightarrow a_2$  means  $a_2$  must follow  $a_1$ . If any of the actions fails, the execution stops.

Constraint predicate (**when** clause in the **OverloadHandler** policy example) must evaluate to *true* for the policy to apply. It can be either time or state based:

- Subject/target state – constraints based on the object state as reflected in terms of attributes at the object interface.
- Action/event parameters – constraints based on event parameter values in obligations.
- Time – constraints, which specify the valid periods for the policy. Usually in the form: **Time.methodName(parameters)**.

Administrator controls overall policy enforcement via policy control object. The object evaluates subject set (a set of relevant enforcement agents i.e. management agents) and passes to each subject a state of the policy enforcement object. **subject** clause of the obligation policy specifies the agents to which the policy must be distributed for interpretation and enforcement.

Policy enforcement object is created at the agent as a CORBA valuetype. State information (transferred from PDP) and enforcement object class information (obtained from Policy Enforcement Class Service) are necessary for the creation.

Two important services: domain and policy repository service are omitted from the figure 4, and are not described here because their functionality and role are standard in most policy based management systems [8].

Each enforcement agent type in management system gets specifications of its own implementation classes for policy enforcement. These classes can be tailored to various agent types (e.g. SNMP, CMIP or CORBA based agents). Moreover, each policy enforcement object can be created in different state. It is well known that object's state direct its behaviour [6] and such mechanism can efficiently be used to modify and adjust the object to the specific management situation.

Beside policy enforcement object creation, enforcement agent has to control it. The control comprises connection and disconnection of the event service which has to dispatch event information. Event types are specified in the **on** clause of the policy specification. Event occurrence triggers obligation method invocation on the policy enforcement object. Obligation method evaluates all the conditions specified in the **when** clause if any. If the evaluation results to *true*, actions specified in the **do** clause of the policy specification are performed on target set. In the context of the obligation policies, target set represents managed objects, and optionally some objects in the subject set (i.e. management agents).

#### A. CORBA Valuetypes

The "object by value" concept was introduced in the CORBA 2.3 standard [4] through the valuetype construct. It enables passing an object by value rather than by reference. An essential property of the valuetypes is that their implementations are local, and their use does not involve the Object Request Broker. This means better performance of the policy enforcement object operation. Moreover, local creation with state information transfer and implementation class service provides flexible and efficient mechanism for adaptable management agent development.

In the figure 6, IDL specification of the policy enforcement object as a CORBA valuetype is shown. The interface `StandardControl` represents standard messages that enforcement agent of any type communicates to the valuetype object. Variables relevant for state (behaviour) of the enforcement object are encapsulated in the `StateSpec` structure.

```
module obligation_policy {
    interface StandardControl {
        connectEventService(in EventSet e);
        disconnectEventService();
    };
    struct StateSpec {
        // state variable specification
    };
    valuetype PolicyEnforcement supports
        StandardControl {
        private StateSpec state;
        factory create(StateSpec s);
    };
};
```

Figure 6: Policy Enforcement valuetype: CORBA IDL

Flexibility and strength of the proposed approach lies in the combination of transferable state and policy enforcement object's class specification (Fig. 7). Such

mechanisms give us the opportunity to dynamically tailor the enforcement agent functionality.

```
package obligation_policy
class PolicyEnforcementImpl extends
    PolicyEnforcement implements . . . {
    /* . . . means whatever interfaces
       enabling interactions whit real
       managed envrionment and real
       enforcement agent
    */
}
```

Figure 7: The policy enforcement object class

#### IV. CONCLUSION

The motivation for the work described in this paper is the need for better understanding how to construct more adaptable management agents. Standard policy-based management architecture powered by the "object by value" technologies seems to satisfy adaptability requirements that are set on the modern telecommunications and systems management agents. This work is not finished, and in the future will be focused on a more detailed evaluation of the object by value mechanisms with respect to the following criteria: importance and implications of the policy object's state information and the adaptability to various types of management agents and its environments.

#### V. REFERENCES

- [1] Damianou N. C., "A Policy Framework for Management of Distributed Systems", Ph Thesis, Faculty of Engineering, University of London, February 2002.
- [2] Kunstic M, Pogacnik F, Sandri N, "A Global Model for Solving the Management Problems in Telecommunication System", Conf. on Communication Networks and Distributed Systems Modeling and Simulation (CNDS'97), *Proceedings of the Conf.*, pp.17-22, Phoenix, 1997.
- [3] *Object Constraint Language Specification*, UML v.1.1, Rational Softw. Co., 1997.
- [4] OMG, The Common Object Request Broker – Architecture and Specifications, Revision 2.4.2, OMG Document formal/01-02-01, Object Management Group, February 2001.
- [5] Randic M, *The Formal Model of Distributed Object-Oriented Software for Telecommunications System Management*, Ph Dissertation (in Croatian lang.), ZZT, FER, University of Zagreb, 1998.
- [6] Randic M, Kunstic M, Blaskovic B., *Information Modeling of Applications Using Mobile Management Agents with Extensible Behaviour in Run-time*, 10<sup>th</sup> Mediterranean Electrotechnical Conference, MeleCon, Cyprus, May 2000.
- [7] Sloman, M. S., *Policy Driven Management for Distributed Systems*, Journal of Network and Systems Management, vol. 2(4), pp. 333-360, December 1994.
- [8] Yavatkar R., Pendarakis d., Guerin R., A Framework for Policy-based Admission Control, RFC 2753, January 2001.