A2A: An Architecture for Autonomic Management Coordination

Alexander V. Konstantinou[1] and Yechiam Yemini[2]

[1] IBM T.J. Watson Research Center
19 Skyline Drive, Hawthorne, NY 10532, USA
`avk@us.ibm.com`
[2] Columbia University
New York, NY 10027, USA
`yemini@cs.columbia.edu`

Abstract. A central challenge of autonomic systems is how to discover, monitor, analyze and control configuration data to assure operational integrity. Current architectures for configuration data management focus on federating repositories that are loosely synchronized, and do not offer autonomic coordination services. We present A2A, a novel autonomic peering architecture which delivers a unified and consistent view of actual element configuration for autonomic systems and managers, and provides synchronization primitives enabling policy coordination and mediation. We discuss the different synchronization semantics and protocols used by systems and managers to access and manipulate configuration data stored in a distributed Modeler. We show how dependent or conflicting policy actions can be automatically detected, correlated and brought to mediation. The A2A architecture has been partially implemented in a large prototype system that has been successfully demonstrated in security, network configuration, and active network applications.

1 Introduction

In traditional manager-agent architectures [1–5] configuration data is stored in a Configuration Management Database (CMDB). The CMDB is populated by discovery agents which loosely synchronize its data with the actual configuration state of the systems being managed. In a typical ITIL [6] process managers trigger discovery to populate the CMDB and then compare the authorized (expected) configuration against the actual (discovered) configuration. When unauthorized configuration drifts are detected a Request for Change (RFC) is generated to update the primary local configuration of the effected systems. At the next discovery cycle these changes are picked up and verified so that the RFC ticket can be closed.

An autonomic system is one that is self-configuring, self-optimizing, self-healing and self-protecting. The manager-agent approach to configuration management is particularly ill-suited for such systems. The self-management actions of autonomic systems quickly invalidate the discovered information stored in

the CMDB, complicate the determination of what represents an unauthorized drift, and can immediately override any RFC actions which conflict with their policies. Autonomic system policies are often driven by changes in the state of other systems. Therefore a central challenge of an autonomic system is how to discover, monitor, analyze and control configuration data to assure operational integrity. An autonomic system may be misled by outdated configuration data in the CMDB. It may thus be unable to synchronize its configuration change transactions with the underlying managed systems and the models of their state stored in the CMDB. Furthermore, the autonomic system may be unable to coordinate its configuration changes with similar actions by other autonomic systems. These various potential inconsistencies between the state of the system, the CMDB model of this state, and actions by autonomic components may lead to significant self-configuration errors and loss of operational integrity.

In this paper we introduce a novel peering architecture for autonomic configuration management. In our A2A architecture the traditional roles of manager and agent are unified as peers accessing a distributed configuration model. Managed elements maintain a local object repository (Modeler) which is accessed through a unified set of transactional local or remote interfaces. In this approach, even though management functions may continue to be distributed across local agents and remote managers, their interactions can be monitored, coordinated and mediated. Recent works [7–9] have considered applications of peering technologies to support discovery and distribution of management data. The primary contribution of this paper is in extending peering semantics beyond discovery and access of shared data, to support safe distributed configuration access for policy verification and configuration propagation.

In Section 2 we present our A2A autonomic peering architecture in relation to the current manager-agent architecture. In Section 3 we elaborate on the semantic coordination functions of the A2A Modeler. In Section 4 we link the architecture to patterns for building manageability by design, and declarative policies supporting static analysis. In Section 5 we briefly discuss the NESTOR prototype implementation. Section 6 discusses related work, followed by some brief conclusions in Section 7.

2 A2A Peering Architecture

We will use a simple configuration propagation example to describe the architecture and operations of the A2A peering architecture and contrast it with current manager-agent solutions. The propagation in our example will be the port and address configuration of a web-server to its clients. As depicted in Figure 1 the operating system *C1* hosts an HTTP server *B1*, requiring client *A1* to configure the host and port through which they access the service.

Consider first a federated solution based on the proposed Configuration Management Database Federation (CMDBf)[5] architecture shown in Figure 1. A Manager, depicted at the top left, needs to enforce the configuration policy by propagating configuration changes on the web-server to its respective clients. The

Manager must first discover the clients and servers whose configurations may be mismatched. This is accomplished through a federated CMDB (1). The CMDB, in turn, accesses Management Data Repositories (MDRs) to obtain the required configuration data. The MDR collects configuration data from the managed elements (*A1*, *B1* and *C1*) through various Discovery Agents (3) which use a variety of protocols (4) (WSDL, SNMP and SSH) to discover managed elements and extract the configuration data from their internal configuration repositories. Once the Manager obtains the configuration data, it needs to propagate the policy changes to the managed elements via respective Provisioning Agents (6) which may in turn utilize a variety of protocols and subsystems to effect configuration changes at the client *A1*.
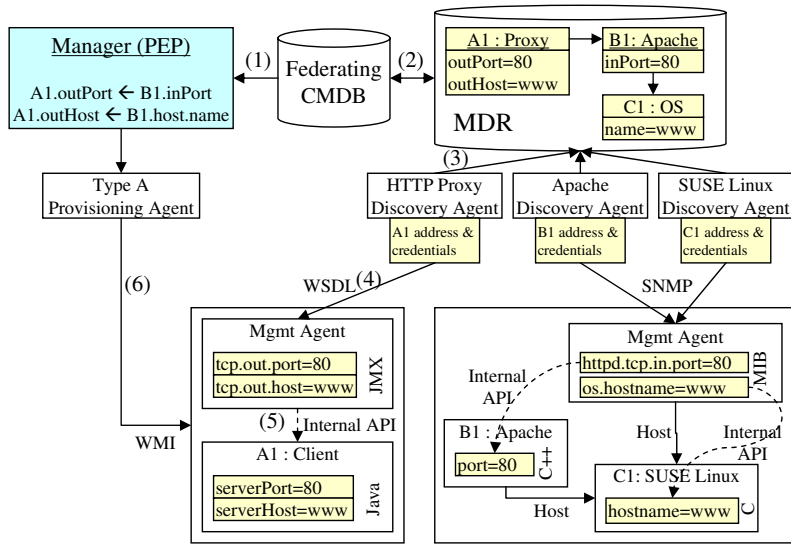

Fig. 1. Manager-Agent Architecture Example

It is simple to see that this fragmented process can be highly sensitive to the flow of configuration data and changes, resulting in complex failures. The MDR data may be outdated, leading to erroneous Manager decisions and actions. Synchronizing the data collection activities of the Discovery Agents with the Manager decision processes and the configuration changes by the Provisioning Agent is impractical. The configuration changes effected by the Manager may conflict with concurrent configuration changes by other Managers. Worse, configuration changes by the Manager may conflict with internal configuration control processes of the managed elements. The configuration changes by the Manager may trigger a cascade of additional configuration changes by other Managers and elements. Such cascading can lead to cyclical loops of changes and non-deterministic operational behaviors, which may trigger problem management activities. For example, a self-configuration action on the client to use a well-known port may be triggered if its port is changed and immediately override

the value. On the next discovery cycle the Manager would re-detect the drift and reconfigure, leading to a cycle. It is thus practically impossible to establish semantically consistent self-configurations through a manager-agent architecture.

Our A2A autonomic peering management architecture, considered first in [10] is depicted in Figure 2. The A2A architecture organizes autonomic peers, typically, but not necessarily embedded in managed elements, into a two-layer architecture. At the bottom layer, a distributed object Modeler, similar to the CMDB, provides a consolidated element data repository, including configuration, relationship, state and performance attributes as well as their behavior events. Modeler objects are instances of classes declared in a unified management Model, such as CIM [11]. The Modeler provides a local North-South API to transactionally access and manipulate the managed data, and subscribe for events. This enables the management layer, above, to access a unified data model, interpret its behavior and activate autonomic control functions. The remote East-West API is used to federate with other Modelers to support access to configuration information in remote object repositories. Existing P2P management protocols [7–9, 12] can be used to affect discovery and DHT-based sharing of distributed configuration data.
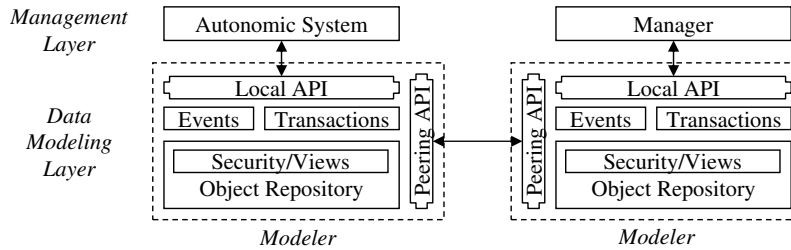

Fig. 2. A2A Architecture

An A2A architecture removes the primary storage of a system's configuration from the system's code and maintains it in the Modeler. It then offers a common set of transactional semantic interfaces to the Modeler enabling peers to access and manipulate configuration data and coordinate these transactions among them. This creates a peering relationship among all processes, both local and remote, wishing to transact with element configuration data based on common semantic abstractions of configuration objects and their manipulations.

We use Figure 3 and the scenario of enforcing the simple propagation policy of Figure 1 to illustrate our A2A mechanisms. The first key difference is that the managed elements do not store their configuration internally. Instead they retrieve and update it in a transactional manner from their local Modeler. For example, the *B1* Apache HTTP server system will bind its server socket to the port based on its configuration in the Modeler. Should that configuration change, it subscribes for an event that will trigger a rebinding of the socket. In the A2A architecture management functions can be flexibly distributed among systems. The function to maintain consistent configuration between the client

A1 and the web-server *B1* can be evaluated and maintained on the client system itself as a self-management action, or in an external manager. In either case, the verification will be associated with a transaction to change the configuration of *A1* or *B1* and synchronously ensure that any change is propagated to the client. The propagation policy may be programmed declaratively using an object-spreadsheet language such as OSL[13]. Once the *A1* manager has computed the appropriate policy decisions it will join the update transaction and set the value of the host and port configuration for *A1*. This update will generate a Modeler event, which the system implementation code will trap to trigger a rebinding of the client socket to the correct host and port.
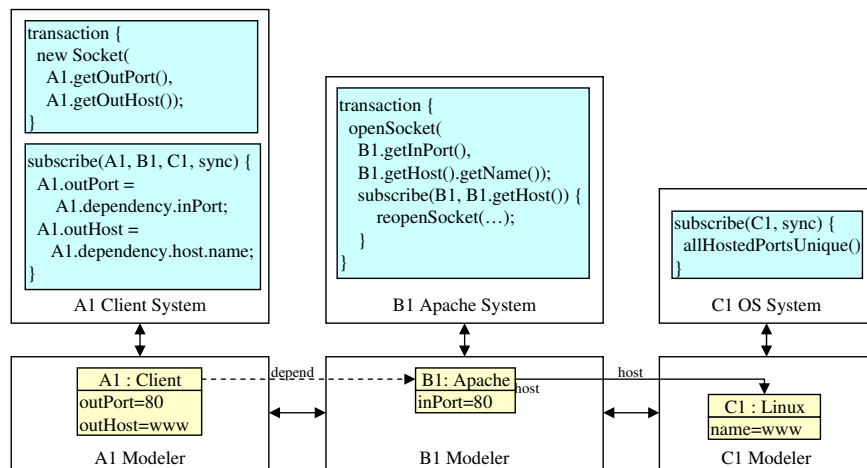
```
transaction {
  new Socket(
    A1.getOutPort(),
    A1.getOutHost());
}
```

```
subscribe(A1, B1, C1, sync) {
  A1.outPort =
    A1.dependency.inPort;
  A1.outHost =
    A1.dependency.host.name;
}
```

A1 Client System

```
transaction {
  openSocket(
    B1.getInPort(),
    B1.getHost().getName());
    subscribe(B1, B1.getHost()) {
      reopenSocket(…);
    }
}
```

B1 Apache System

```
subscribe(C1, sync) {
  allHostedPortsUnique()
}
```

C1 OS System

A1 : Client	B1: Apache	C1 : Linux
outPort=80	inPort=80	name=www
outHost=www		

A1 Modeler   depend   B1 Modeler   host   host   C1 Modeler

Fig. 3. A2A Architecture Example

    This semantic peering architecture for autonomic systems offers several significant advantages over the traditional manager-agent (client-server) organization. It moves configuration data from local internal repositories to a unified distributed modeling layer and eliminates the need to maintain replicated repositories at systems, agents and CMDBs, and to synchronize them across different data models and access protocols. The common transactional semantics, provided by the Modelers, enable direct shared access to configuration data among all interested components, whether local or remote. This permits coordination and synchronization by distributed systems and management components of configuration access and changes. Furthermore, the shared transactional semantics enable checkpointing of configuration states, analysis of failures and recovery. The peering architecture permits flexible scaling and changes of configuration management. New elements, managers, systems components and autonomic policies can be easily joined, or removed involving modular relatively simple changes. Moreover, the peering architecture permits robust operations of autonomic management. The management system can continue to operate under dynamic changes, failures and partitioning of the network. It can effect

autonomic policies that gracefully reduce unavailable services and reconfigure resources to support self-healing of services that may be sustained through the failure. The unification of the traditional roles of manager and element allows management functions to be flexibly distributed at different elements, supporting autonomic behavior. In what follows we describe in details the A2A mechanisms for semantic peering.

3   Distributed Modeler

A central function of the A2A architecture is to enable peer managers to access, share and manipulate distributed configuration data. This function is primarily handled by the distributed Modeler. The Modeler provides (a) a repository of object-relationship data models and meta-models; (b) mechanisms for repository access and manipulations supporting view abstractions and security protections; (c) mechanisms to support distributed transactions by local and remote managers, and (d) publish/subscribe mechanisms to handle events.

We illustrate the Modeler operations using an autonomic policy maintenance scenario depicted in Figure 4. In response to a change in the environment or its policy an autonomic manager (*mgr2*) determines that the *outPort* of *B1* should be changed from 80 to 81. *Mgr2* creates a write transaction (*t2*) to update the state of *B1.inPort* in the Modeler *b1m*. A second autonomic manager *mgr1* has a synchronous event subscription on this property and will join the transaction *t2*. Once *mgr1* has joined, the set operation of *mgr2* returns, and *mgr2* indicates that it is prepared to commit. In parallel, *mgr1* evaluates its own policy which computes a change in *A1.outPort*. *Mgr1* contacts Modeler *a1m* which also joins the transaction *t2*. Modeler *a1m* records the change to *A1.outPort* in the context of transaction *t2*. The change does not involve *mgr2*, and thus the propagation is complete and all parties can vote "*prepared*" to commit. The transaction manager will inform the participants of their joint decision to commit, and collect their acknowledgments.

3.1   Object-Relationship Repository

The use of object-relationship models for the configuration of systems[14] and networks[15] has been widely adopted in industry[16, 17], and applied to autonomic systems[18]. Autonomic object models can encapsulate both configuration as well as performance instrumentation data, as described in [10]. For example, a model for an IP interface can represent both configuration information, such as address and netmask, as well as performance information, such as number and average size of transmitted packets. Relationships express configuration, and hence operational, dependencies between systems. The network data model has been studied extensively in the context of database systems[19]. The A2A object-relationship repository can thus be implemented over proven distributed object-oriented database technologies.
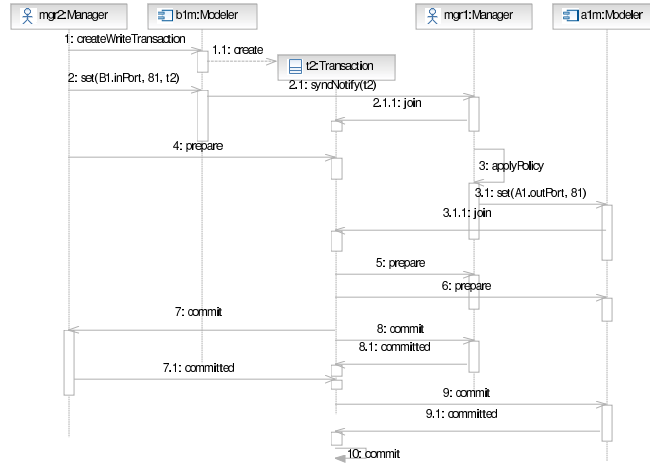
Fig. 4. Configuration Propagation Example

Management repositories model physical and software systems that can be moved, rewired, or fail. For example, the network cable connecting a server to a switch can be moved to another port. A hard drive in a RAID array can fail, reducing the overall reliability of that system. One can therefore consider managed systems as being described by a number of models. One model is that of the *actual* configuration which represents the current state of the system. Another model represents the *authorized* configuration, which represents the expected state of the system. Due to physical reconfigurations or failures, the authorized configuration may not match the actual configuration, and a traditional database rollback may not be feasible. In such cases, it is necessary to enter a mediation process to compute a *desired* state model[20, 21] which represents a new valid actual state. Provisioning of this desired state model may involve automated and manual tasks for reconfiguring existing systems and deploying new ones as reported in [22].

3.2 Distributed Transactions and Events

Autonomic systems and managers must assure semantic consistency of configurations. This requires evaluation of self-configuring policies over a consistent view of configuration. Managers must also coordinate configuration changes used to enforce their policies.

In contrast to traditional DBMS, a range of transactional semantics must be supported to access the different configuration models. The actual configuration model can only be modified by local instrumentation code, and cannot be locked by managers. For example, the instrumentation of a system's Ethernet port will update the MAC address of its peer in the Modeler. Negotiation over the desired state model can be performed atomically, in isolation, and persisted durably, but may not be isolated from concurrent changes in the actual model. For example, in the process of evaluating a security policy based on the MAC address of an

Ethernet port, the connection may be physically severed. Finally, in the process of provisioning a desired state model, uncontrolled threads of change and failures may invalidate design assumptions, or partially configure the desired state.

When policies are evaluated against the actual configuration model, they must be protected against controlled as well as uncontrolled changes. Policy evaluation does not change the actual model, and thus is inherently re-entrant. An optimistic concurrency approach can therefore be taken, whereby the policy is reevaluated if a change is detected during policy evaluation. In the Ethernet port example, the optimistic lock on the configuration of the peer's address will fail, and the policy will need to be reevaluated over the new value. Any number of optimistic concurrency control algorithms can be used for this purpose[19].

Policy violations will involve reconfiguration actions by the autonomic systems and managers. These actions must be coordinated to prevent inconsistencies and propagation loops. For example, a policy to propagate the port of a server to all clients may conflict with a policy to use well-known ports for client communication. In the A2A architecture, coordination is achieved in the context of a distributed transaction over the desired state model. The shared transaction becomes a Space[23] which autonomic systems can query for other systems and managers, their planned configuration actions, and the policies which triggered them. Policy conflicts can be detected, correlated to their source, and mediated. The specific mechanisms used for mediation are beyond the scope of this paper.

The mechanism used to form a shared transaction between interacting autonomic systems and managers is based on a synchronous event service. When policies are verified against the current state using an optimistic transaction, systems and managers can subscribe for changes to the actual or desired state of the objects, attributes and relationships they have accessed. For example, the manager verifying the policy over the use of well known ports will register for changes to the port attribute of client types in all objects of a specific management domain. Event subscriptions can be synchronous or asynchronous. Synchronous event notifications are performed in the context of the transaction which triggered them, and allow subscribers to join in that transaction. In our example, when a management process changes the port of the server, this will trigger a synchronous event to the manager maintaining the server to client propagation policy. Failure to notify a synchronous subscriber is treated as a vote to abort the transaction. Standard publish-subscribe architectures and protocols can be utilized[24] for message delivery.

Arranging for all autonomic managers with policies operating on the same changed data to join a distributed transaction is one aspect of coordination. The second aspect is the distributed transactional protocol employed. The protocol must be resilient to failures of systems. In our example, when the manager enforcing the server to client port propagation policy receives a synchronous event, it joins the transaction. Upon evaluating its policy, it determines that the client port must be changed, and associates additional changes with the transaction. The change to the client port will notify the manager of the well-known client port policy who will recognize a conflict, and identify the policy

which triggered the cascading failure. At that point, the transaction can be aborted, or mediation between the two conflicting policies can be enforced. At any point during this transaction, one of the managers can fail. The transaction protocol must assure consistency and recovery from such failures.

The three-phase commit protocol (3PC) supports distributed coordination with a quorum-based recovery procedure when failures are detected[25, 26]. Figure 5 depicts a modified state diagram for a 3PC participant. Upon a change in the actual model, or the change action of another manager over the desired state, effected managers are joined to the transaction. All managers start in state R. They apply their policies over the respective Modeler repositories concurrently. Each Modeler maintains a read set and a write set for every participant. When read $\rightarrow$ write conflict is detected, it is necessary for the policy which read the overridden value to be reevaluated. We modify the 3PC protocol by adding a new message called *restart*. A restart message to the coordinator with all participants in the R or V triggers a restart message to the participants specified in the restart message. The identified participants in the V state will transition back to the R state and will re-evaluate their policies. Cyclical write propagations or otherwise inconsistent policies leading in write $\rightarrow$ write conflicts can result in restart loops. A transaction coordinator is responsible for detecting such loops and triggering policy mediation.
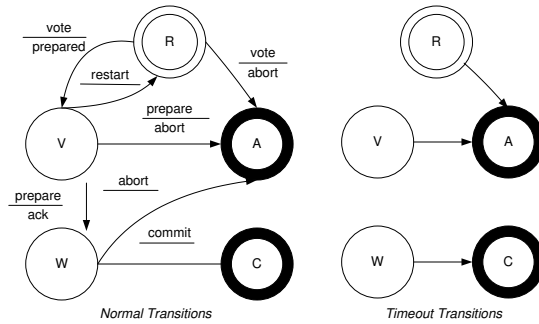

Fig. 5. Restartable 3PC Protocol

The architecture of the Modeler's event and transaction mechanisms supports a variety of optimizations. Profiling of autonomic systems and policies can be used to optimize the evaluation order of interacting policies. The results of conflicting policy mediation can produce composite policies that will prevent future conflicts. Policy cycles across transactions, due to undetected propagation paths, can similarly be correlated, triggering cross-transaction policy mediation.

3.3 Security

Access to detailed system configuration models is a major security concern. Intruders can use such information to quickly identify architectural and system

vulnerabilities. Configuration information can also be utilized in social engineering attacks. Obtaining the ability to change configuration can usually be used to compromise or attack a system. By separating the configuration repository from the system's operational code, the A2A architecture can reduce the risk of unauthorized configuration changes. Traditional database view-based security and policy-based security approaches[27] can then be enforced directly over the network model layer. The A2A architecture can be used to associate role credentials over autonomic system containment and communication paths. For example, Modeler credentials can be communicated over secure application-level communication channels.

4 Manager Layer

In the A2A management layer, autonomic systems and managers operate as peers over a shared configuration model which is accessed through the Modeler described in the previous section. Autonomic systems must be designed and implemented to integrate with the Modeler. As such, all their configuration attributes must be exposed through a meta-model that is instantiated and persisted within the Modeler. Autonomic element instrumentation functions for performance and physical configuration attributes must be separated from the system's autonomic functions. Beyond the instrumentation aspect, and the use of local APIs to access the Modeler, there is no difference between the built-in autonomic functions of a system and those of external managers. The management layer does not impose any additional restrictions on the design of autonomic systems and managers.

The distributed Modeler layer is the foundation of two key autonomic manager technologies on which we have previously reported. JSpoon[10, 13] is a language for integrating the configuration and instrumentation aspects of autonomic agents and managers at design time. JSpoon provides native transactional and event primitives to support patterns of instrumentation and management access. The JSpoon runtime interfaces to the Modeler distributed transaction and event functions to support the language features. The Object Spreadsheet Language (OSL)[13] is a declarative expression language extending OCL[28] to encode configuration propagation policies. By leveraging the A2A Modeler's transactional and event interfaces, a distributed incremental OSL interpreter was built with support for static as well as dynamic propagation path conflict analysis and mediation.

5 NESTOR Prototype

The A2A peering architecture outlined in this paper has been partially implemented in a large research prototype called NESTOR[13]. The prototype includes a custom distributed transactional object Modeler, an object-relationship modeling definition language compiler, an incremental Object Spreadsheet Language (OSL) change rule and Object Policy Language (OPL) constraint interpreter,

adapters for different management protocols and elements, and a rich management graphical editor. The NESTOR transactional model is based on optimistic concurrency control for instrumented configuration attributes, and 2PC for desired configuration state. The R3PC algorithm presented in this paper was not integrated into the platform. NESTOR was developed in two successive versions which provided practical experience with different automation architectural designs. NESTOR has been applied to the management of security in dynamic networks[29], automating configuration of network virtualization[30], instrumenting Active Network Nodes, and configuring a distributed firewall based on security policies[31].

6   Related Work

Several recent works [13, 12] have thus explored distributed architectures for autonomic systems. A distributed management architecture, first proposed at [32], enables autonomic components to assure consistency of their views and actions with the actual states of managed subsystems, by directly manipulating local repositories of these subsystems. The novelty of our approach is that we focus on the safe distribution and transactional manipulation of element configuration over which existing mechanisms for distributed policy enforcement and collection can be layered.

A recent draft specification for a CMDBf federation standard[5] is an attempt to standardize integration based on a federated architecture. In this architecture, configuration data is aggregated in a federated database which is populated by pull or push of data from distributed Management Data Repositories (MDRs). The standard defines a graph-based query language for performing queries over the federated database that can be distributed to the MDRs. The CMDBf architecture does not challenge the basic design goals of the CMDB, and therefore does not address transactional access, synchronization, or provisioning. The A2A architecture refactors the federation and query concepts of CMDBf into a two-layer peering management architecture to support autonomic services.

The challenge of data center management automation has been the subject of a number of previous studies [18, 33]. A common assumption in a number of these studies is that a consistent view of the world is an input to the system. Often, the stated goal of these systems is to enforce centrally defined functional and non-functional policy constraints. Automation is thus introduced at the management layer and assumes that the managed systems are not self-managing or have limited and well defined autonomic functions. Based on these assumptions the manager can plan the changes required over the consistent world view input to provide or optimize some function, and then schedule the provisioning of these changes. The A2A architecture is an enabling technology for these studies, because it provides an essential mechanism for obtaining a consistent world view. More importantly, it will also support distributed approaches to automation using cooperating self-managed and self-healing processes.

A number of recent studies have focused on the application of P2P technologies to management [7–9, 12]. Their emphasis has been on P2P discovery and DHT-based sharing to distribute data across multiple managers for scalability. The use of P2P discovery techniques has been incorporated in our A2A architecture. The approach to creating uncontrolled copies of configuration data, however, exacerbates the synchronization challenges we have identified as a key challenge for autonomy. Our architecture specifies detailed synchronization primitives which can provide safe access to distributed peers, and enable them to join in order to negotiate over their policies.

Autopilot [34] is an example of a new generation of data center management systems. These systems are characterized by a small number of vertically engineered applications that were designed to scale to hundreds of thousands of homogeneous software and hardware instances. A fault-tolerant centralized device manager receives information from data center systems and monitors which is used to update a strongly-consistent current state data model. The device manager then uses a set of manually determined policies to compute the desired data center state model. Management services and systems respectively provision and reconfigure based on their determined desired state. The architecture supports weak-consistency of the deployment state during provisioning. Our A2A architecture can be viewed as a first attempt at bridging the manager-agent and vertical types of data centers, supporting both strong and weak consistency models.

7   Conclusions

The current trends towards delivery of software as a service are shifting management complexity from client systems into mission critical data centers which are rapidly evolving and enlarging. Within the data center there have been two fundamental reactions to these challenges. Enterprise data centers have focused on applying ITIL application-lifecycle technologies capturing existing best-practice workflows over a shared CMDB. These workflows tend to be static and human task oriented, admitting limited automation at predetermined points. The other reaction comes from a new generation of massive Internet applications, such as web-search, which were designed in a vertical manner, integrating autonomic policies for deployment, monitoring, failure recovery and migration.

These two diverging data center architectures present different challenges to the adoption of autonomic technologies. At the Enterprise-level, the proliferation of heterogeneous services with complex hosting and connectivity dependencies, accessed via fragmented management functions creates a high-risk environment for autonomy. Autonomic technologies will not be adopted without clear controls over what can be changed, based on what information and by whom. The current approaches towards weekly consistent CMDBs, while a big step forward, will not deliver a platform that is safe for autonomy. Our proposed A2A semantic peering architecture puts an emphasis on consistency and coordination, which are the

cornerstones for effective autonomic technology adoption in the Enterprise data center.

The challenges posed to autonomy by massive Internet application data centers are less well understood at this point. We expect that the main challenge will be in composing the autonomic functions of different vertical applications[35]. The widening offering of services, from search to mail, and now productivity applications will challenge the vertical structure of such centers. The A2A architecture can provide a space for safe mediation of the interacting autonomic functions of these vertical semi-autonomous applications. Furthermore, it promises to unify the management architecture across types of applications, which will become increasingly important as their domains begin to overlap.

Acknowledgments

References

1. ISO: OSI basic reference model - part 4: Management framework. Technical Report 7498-4, ISO (1989)
2. Case, J., Fedor, M., Schoffstall, M., Davin, J.: A Simple Network Management Protocol (SNMP). Technical Report RFC 1067, IETF (1988)
3. Martin-Flatin, J.: Push vs. pull in web-based network management. In: Integrated Management. (May 1999)
4. Schonwalder, J., Pras, A., Martin-Flatin, J.P.: On the future of internet management technologies. IEEE Communications Magazine **41**(10) (2003) 90–97
5. CMDB Federation Workgroup: CMDB federation CMDBf. Technical report (2008)
6. Office of Government Commerce: The Official Introduction to the ITIL Service Lifecycle Book. The Stationery Office (August 2007)
7. Zach, M., Fahy, C., R.Carroll, Lehtihet, E., Parker, D., Georgalas, N., Nielsen, J., Marin, R., Serrat, J.: Towards a framework for network management applications based on peer-to-peer paradigms the celtic project madeira. In: IEEE NOMS. (April 2006)
8. Tang, C., Chang, R.N., So, E.: A distributed service management infrastructure for enterprise data centers based on peer-to-peer technology. In: IEEE Services Computing Conference (SCC). (September 2006)
9. Granville, L.Z., daRosa, D.M., Panisson, A., Melchiors, C., M.J.B.Almeida, L.M.R.Tarouco: Managing computer networks using peer-to-peer technologies. IEEE Communications Magazine (2005)
10. Konstantinou, A.V., Yemini, Y.: Programming systems for autonomy. In: IEEE Autonomic Computing Workshop (AMS 2003), Seattle, WA (June 2003)
11. Distributed Management Task Force (DMTF): Common Information Model (CIM) specification. Technical Report Version 2.2, DMTF (June 1999)
12. Kamienski, C., Sadok, D., Fidalgo, J., Lima, J.: On the use of peer-to-peer architectures for management of highly dynamic environments. In: 4th IEEE Int. Conf. on Pervasive Computing and Communication. (March 2006)

13. Konstantinou, A.V.: Towards Autonomic Networks. PhD thesis, Columbia University (October 2003)
14. Sloman, M.: Management for open distributed processing. DCS **1**(9) (1990) 25–39
15. Dupuy, A., Sengupta, S., Wolfson, O., Yemini, Y.: Netmate : A network management environment. IEEE Network Magazine (1991)
16. DMTF: Common Information Model (CIM). Technical report, DMTF (2006)
17. W3C: Service Modeling Language, version 1.0. Technical report (2007)
18. Yemini, Y., Konstantinou, A., Florissi, D.: NESTOR: An architecture for NEtwork Self-managemenT and ORganization. IEEE JSAC **18**(5) (2000)
19. Elmasri, R., Navathe, S.B.: Fundamentals of Database Systems. 5 edn. Addison-Wesley (2006)
20. Eilam, T., Kalantar, M., Konstantinou, A., Pacifici, G., Pershing, J., Agrawal, A.: Managing the configuration complexity of distributed applications in internet data centers. IEEE Communication Magazine **44**(3) (2006) 166–177
21. Arnold, W., Eilam, T., Kalantar, M., Konstantinou, A.V., Totok, A.A.: Pattern based SOA deployment. In: ICSOC. (September 2007)
22. El Maghraoui, K., Meghranjani, A., Eilam, T., Kalantar, M. Konstantinou, A.: Model driven provisioning: Bridging the gap between declarative object models and procedural provisioning tools. In: 7th Int. Middleware Conference. Volume 4290 of LNCS., Springer-Verlag (2006) 404–423
23. Gelernter, D.: Generative communication in linda. ACM Trans. Program. Lang. Syst. **7**(1) (1985) 80–112
24. Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.M.: The many faces of publish/subscribe. ACM Comput. Surv. **35**(2) (2003) 114–131
25. Skeen, D.: A quorum-based commit protocol. In: 6th Berkeley Workshop on Distributed Data Management and Computer Networks. (February 1982)
26. Keidar, I., Dolev, D.: Increasing the resilience of atomic commit, at no additional cost. In: PODS '95: Proceedings of the fourteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, ACM (1995) 245–254
27. Sloman, M., Lupu, E.: Security and management policy specification. IEEE Network (2002)
28. OMG: Object Constraint Language specification (OCL). Technical Report ad/97-08-08 (version 1.1), Object Management Group (OMG) (September 1, 1997 1997)
29. Konstantinou, A.V., Yemini, Y., Bhatt, S., Rajagopalan, S.: Managing security in dynamic networks. In: USENIX Lisa. (1999)
30. Su, G., Yemini, Y.: Virtual Active Networks: towards multi-edged network computing. Computer Networks **36**(2/3) (2001) 153–168
31. Burns, J., Gurung, P., Martin, D., Rajagopalan, S., Rao, P., Rosenbluth, D., Surendran, A.: Management of network security policy by self-securing networks. In: (DISCEX II), Anaheim, California (2001)
32. Goldszmidt, G., Yemini, Y.: Distributed management by delegation. In: The 15th Int. Conference on Distributed Computing Systems, Vancouver, BC, IEEE (1995)
33. Eilam, T., Kalantar, M., Konstantinou, A., Pacifici, G.: Reducing the complexity of application deployment in large data centers. (2005)
34. Isard, M.: Autopilot: Automatic data center management. Operating Systems Review **41**(2) (2007) 60–67
35. Konstantinou, A., Eilam, T., Kalantar, M., Totok, A.A., Arnold, W., Snible, E.: An architecture for virtual solution composition and deployment in infrastructure clouds. In: 3rd Int. Workshop on Virtualization Technologies in Distributed Computing (VTDC). (June 2009)