J Internet Serv Appl (2012) 3:107–115
DOI 10.1007/s13174-011-0047-6

SI: FOME - THE FUTURE OF MIDDLEWARE

Towards application driven security dashboards in future middleware

Wouter Joosen · Bert Lagaisse · Eddy Truyen · Koen Handekyn

Received: 4 November 2011 / Accepted: 11 November 2011 / Published online: 29 November 2011

Abstract Contemporary middleware must facilitate the customization of a built-in services framework, such that non-functional requirements emerging from the engineering process are met. This must be achieved by facilitating adaptation and selection of appropriate services without carrying the load, footprint, and overhead of a bloated system.

We illustrate the concept and approach with an example in the domain of security engineering of a large scale, internet based application in the domain of online document processing. In addition, we sketch why such an approach cannot only yield the desired variants of middleware security services, but also application-driven security dashboards, i.e. the tools to monitor and manage the actual security environment. The resulting research findings plead for a research agenda that revisits reflection and that enables model-driven software techniques to be used in the just-in-time generation of co-existing middleware variants.

Keywords Middleware · Security · Adaptive middleware · Monitoring · Software-as-a-service

W. Joosen · B. Lagaisse (✉) · E. Truyen
DistriNet, Dept. Computer Science, KULeuven,
Celestijnenlaan 200A, 3001 Heverlee, Belgium
e-mail: bert.lagaisse@cs.kuleuven.be

W. Joosen
e-mail: wouter.joosen@cs.kuleuven.be

E. Truyen
e-mail: eddy.truyen@cs.kuleuven.be

K. Handekyn
UnifiedPost SA, Avenue Reine Astrid 92A, 1310 Terhulpen, Belgium
e-mail: koen.handekyn@unifiedpost.com

1 Introduction

Software systems and applications have become increasingly distributed, complex, and dynamic, for example because of interactions and dependencies with the physical world that arise from various types of sensors—in the so-called *Internet of Things*. Another driver is the increasing interoperation between software artifacts from multiple service providers

This evolution continues and the dynamics, both within the software systems as well as in the environment in which systems and applications are deployed drive the need for flexible modifications, preferably statically, yet by necessity often at run time. Also, multiple software qualities—for instance performance, security, reliability, and availability—must be obtained and preserved during the life time of such dynamic distributed software systems. In fact, the required software qualities can typically not be delivered without the software system being adapted. These observations account for many concerns or properties of a distributed system, this paper zooms into security services in future middleware.

For an example of the security requirements of next-gen middleware, consider the emergence of web 2.0. Complex access control models and privacy controls have become the newest security features. Similarly, the security requirements of ubiquitous electronic services have become more complex, such as data protection, digital signature requests or non-repudiable transactions. In the best case scenario, these complex security services are built directly into the application, but even this is exceptional. The increasing trend of out-sourcing service creation and deployment to multiple external service providers (cloud providers of various natures), is raising the bar once again.

This paper assumes an up-to-date role for middleware starting by analyzing the complexity of future distributed environments in terms of customization, co-existing variants, and management. Our first contribution is in examining the role and needs of middleware in such a complex context; we broaden the scope of middleware beyond the traditional (and essential) development activities. Our second contribution is in presenting a real world and actual case study that represents these needs. Our third contribution is the illustration of the research needs in the specific context of security services. We add important pointers to relevant research that may inspire the research community going forward.

The remainder of the paper is structured as follows. We propose our vision on the future of (security) middleware in Sect. 2. In Sect. 3, we introduce the case study that we have been conducting. Section 4 elaborates on the research challenges for security middleware. We illustrate these challenges in Sect. 5 and present an overview of the envisioned approach. In Sect. 6, we highlight state-of-the-art and identify the needs. We propose research directions in Sect. 7 and summarize in Sect. 8.

2 Vision

Middleware traditionally is a hybrid between development and runtime support. Development support must focus on customization that is more cost effective in terms of man power. On top of that, runtime support must enable monitoring and management in terms of high-level stakeholder-centric representations of the run-time, requiring a minimum of effort in terms of man power. Within the scope of this paper, we illustrate both challenges in the context of security middleware.

To achieve trustworthy cloud-based internet applications, the security middleware must offer appropriate support (1) to support large-scale customization of security policies and the overall security subsystems, (2) to enable optimization of the run-time characteristics of stakeholder-specific configurations, and (3) to offer appropriate dashboards for security monitoring and management for the broad range of stakeholders that are involved in SaaS applications: tenants, application provider, platform provider, customers of tenants.

Such features can build upon reflective facilities of middleware, as well on as model driven techniques. While reflection has focused on implementation-level reification of development-time and runtime concepts, the future needs *higher-level abstractions* such as architectural representations that cover development-time and runtime that address domain-specific concepts such as security policies, as well as application concepts such as business objects. The gap between the required customization and monitoring abstractions on the one side, and the state-of-the-art customization and management facilities on the other side still causes a lot of manual implementation effort. To manage the presented complexity, we need to leverage on model driven techniques to enable automated transformations between abstractions as well as to ensure consistency between development-time and runtime models.

We focus on the importance, but also on the challenges of reflective technologies on which such support for customization, monitoring, and management can build. The two main premises of reflective middleware—inspection and customization—are more important and even more challenging than ever to achieve cost effectiveness in terms of development effort as well as performance.

3 Case study

Our case study is an online distributed software system for document processing and document management. The envisaged application supports the creation, the generation of layout, the business specific processing and the storage of business documents (for example invoices, status reports, work orders, account information, checkbooks, shares, bonds, etc.) for various organizations with a large amount of customers and end users who need to use fractions of these sets of documents. The organizations mentioned above are the main clients of the provider of this distributed document management system. A car leasing company is a good example; they have large corporate customers whose employees are individual end users of the system (alongside appointed members of the HR and accounting department of the given customer).

The distributed document processing and management system is typically deployed as a software service (in the cloud, from the client's perspective). The core of the software system therefore is a multi-tenant platform for e-document production and presentation. Various organizational customers (called tenants) can use the platform to create, store, and send out large amounts of personalized documents to their end users.

The high-level architecture of the platform (see Fig. 1) consists of the following set of sub-systems: a production environment, a document presentation environment, a set of input channels, a document store, and a set of output channels—which we all briefly discuss below.

The input channels Customers and their respective end users primarily use the document management service by using multiple input channels. The customer organizations deliver raw data potentially using a multitude of formats. These data sets contain information such as customer names, billed items, and other customer and other organization specific data.

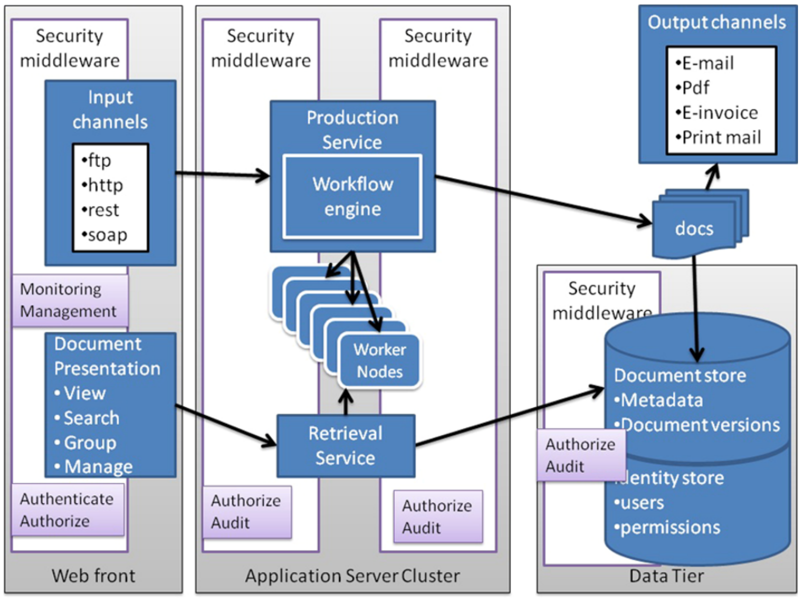

Fig. 1 Security requirements, architecture, and middleware

The production service The central part of the platform is a document production service with a workflow engine that supports the execution of work flows for generating layout, for validation and verification and for processing business documents in a customer-specific way. One type of service can be seen as a script that defines the flow of actions leading to the correct processing of the documents from a particular organization.

Worker nodes The workflow engine executes a sequence of document processing actions. Each action is to be executed by a so-called worker node of a particular type. Example services include services that merge documents, that export content to a database, that append additional notes in a specific format, that generate and verify signatures, etc. These services might require specific hardware that must be supported by the physical node that is executing a given worker. For example, signing hardware may be needed; sometimes specific software may be required that is running on few nodes because only a limited set of software licenses are available. Clearly, there are specific deployment constraints for each kind of service.

The *output channels* provide multiple delivery channels for the delivery of documents to customers, end users and third parties: e-mail, e-mail with specific attachment, e-invoice services, printed mail, etc.

The *document store* keeps track of the initial documents and all associated meta-data, the produced output, and the history (log) of the production process.

In a web-based *document presentation* interface, the tenants and their customers can manage the documents that have been generated on their behalf. This provides users with core services for utilizing and organizing a document archive: search capabilities, labeling features, etc. These management operations are executed by a retrieval service, which can also use the above mentioned worker nodes to process some heavy operations in the background, such as the aggregation of a specific and large selection of documents into a print batch.

A key task of the overall platform is to manage complex security models for defining the relationships between users, customers, clients and documents, for defining privileges and permissions, on generic and specific actions that view and manipulate documents. This requires security solutions to enforce fairly complex security policies, for example with regards to authentication, authorization, and audit requirements. The nature of these security requirements is not new: authentication and authorization have been around for a while, yet the complex relationships and interdependencies of a multi-tenant setting, including the required per-tenant variants and customizations, have not been encountered before.

4 Challenges for security middleware

The growing success of many SaaS applications—such as this (real-life) document management platform—raises many challenges that must be addressed. These challenges relate to the realization of many non-functional requirements such as availability, performance, and security. These challenges also align with the traditional and dual role of middleware as being both an environment that offers development support (application creation), as well as deployment (monitoring and management of applications). Figure 1 shows a conceptual sketch of the application architecture that is surrounded by security middleware, and policies for authentication, authorization, and audit to ease the development and deployment pain of the SaaS provider.

In our experience, the following problems must be addressed and preferably be supported at the level of middleware in order to optimize the man power effective creation and management of these new types of application platforms. We further characterize the challenges:

– *Large-scale customization of security polices and (sub) services:* Next to the basic authorization policies enforced by the SaaS provider, there is an additional need to implement, configure, deploy, manage, and monitor tenant-specific variants of the security policies. This customization can include tenant-specific security policies with regards to non-repudiation and audit trails, or tenant-specific authorization requirements. In effect, such new policies may trigger the extension and/or customization of security functionality for the sake of a tenant specific version of the software application service. Even though a generic base-level security architecture will exist, still variants must be composed to meet customer expectations. This should be performed efficiently.
– *Optimization of the run-time characteristics of tenant-specific configurations:* at deployment time, the chosen and adapted security middleware can still be implemented in various ways. For example: in case of an attribute based authorization service, one can push or pull the necessary attributes to a policy enforcement point, one can enable caching of such attributes at the risk of being slightly outdated when making decisions, and one can optimize the underpinning communication and routing strategy to improve the total cost of communication. In other words: complex systems—even when the security services are given and not evolving—can and should be optimized to deal with performance and availability needs of the security service itself.
– *Consoles for monitoring and management:* Customization and optimization should be implemented at least before the activation of a tenant specific service. However, as requirements may change dynamically, and as the system's load, risks, and execution context will be highly dynamic, these customizations and optimizations should ideally be possible at run time, also. This obviously requires management support.

5 Toward security dashboards: an example

We introduce the notion of a *dashboard* to refer to a subsystem that enables the administrators and business decision makers (product managers, security officers, risk managers, etc.) to observe the behavior of the services at run time, and to enforce adaptations according to the observed needs. It should be noticed that such a dashboard must be made available to the SaaS provider, as well as to their clients, and ultimately to the customers of these clients. Dashboards are in their own turn customized to the specific types of users. Let us now illustrate why all of the above is needed by zooming into one specific security requirement: authorization.

Attribute based access control has been studied and applied for years, with the growing popularity of web services, XACML has emerged to become a de facto standard for practical access control. The underpinning reference model is a reasonable basis for describing authorization solutions at a design level. There are 4 key logical entities in XACML—and in the underpinning reference model.

1. PDP (policy decision point): the entity that evaluates the applicable policies and makes the authorization decision
2. PEP (policy enforcement point): the entity that guards a protected entity, that intercepts requests, and enforces the decisions made by the PDP
3. PIP (policy information point): the entities that act in the role of source of attribute values that are relevant to the PDP
4. PAP (policy administration point): the entity that offers an interface to manage security policies.

Figure 2 shows an extended picture of our application case study; various instances of the authorization (sub)services are included. This shows the omnipresence of just one of the major security services that should be integrated in the middleware. Moreover, the actual security policies and security needs will differ (1) based on varying security functionality for each of the tenants, and (2) based on varying ways to implement and configure the security run time system. Examples of the first cause include:

– The necessity of audit trails: an audit trail might be necessary for tenants in the domain of the financial industry, but not for the document flow of all possible business.
– The authorization model of the tenant: e-health documents can be managed based on the role of the end-user (e.g. hospital administration), while bank documents may be purely managed based on ownership.
– The level of intrusiveness of authorization and audit: does one authorize and audit all calls throughout the entire execution flow, from the web tier over the application tier to the data tier? This is often required; sometimes authorization on the web front is enough though.

Examples of the second cause include:

– The number of deployed instances of PDP, PEP, and PIP, as well as the locations of deployment should be customizable.
– The communication strategy between the deployed PDPs, PEPs, and PIPs can vary: pushing and/or pulling information attributes is one aspect; batching information before distributing a larger set of attributes is another example.

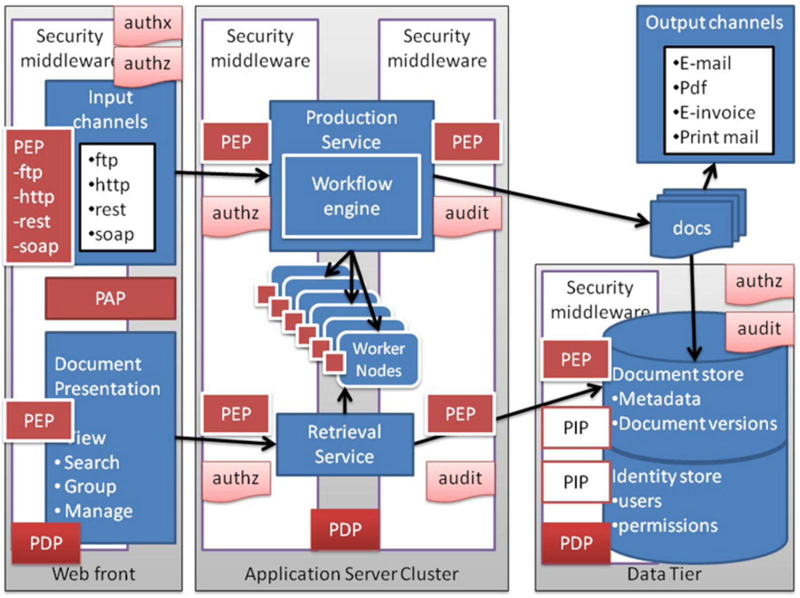

Fig. 2 Deploying the logical entities of XACML

– In addition, PEPs and PDPs caching strategy for information attributes provided by the different PIPs.

In summary, we should be able to support controlled modifications of the security logic, as well as of the configuration of its execution environment. Therefore, application dashboards are needed, and in light of the specific focus of this paper, security dashboards in particular. These dashboards should enable and simplify the management of applications, middleware, as well as variations of both as needed by application providers, tenants, and customers.

The previous customizations with regards to policies, deployment, and communication can vary for each tenant, and must therefore *co-exist at runtime*. The SaaS provider must be able to manage all these co-existent variations in the security middleware, and each tenant must be able to manage their own set of required security services. For example, consider the two co-existent variations as illustrated in Fig. 3. Next to the basic authorization policies of the platform owner, there are two tenants (assumed from the financial industry) with their own policies and associated deployment strategy:

– Role based authorization for a first tenant, when entering the application tier, as well as when entering the data tier.
– Owner-ship based authorization and deep auditing for a second tenant, throughout the full control flow. Because this tenant delivers check books and printed bonds to its customers, each produced document must be traceable in the audit trail, and each access to this document must be audited. In the security dashboard, each produced document can be tracked, and each delivery of the document can be tracked.

6 State-of-the-art and related work

The requirements that have been discussed in the previous section are not entirely new; in fact, one could argue that many ingredients and elements have been on the research radar for a while, yet many of the known building blocks, methods, and techniques are challenged in current and future settings. In addition, integrating multiple research results in holistic solutions will remain a challenge for a while.

It would be impossible to exhaustively discuss all the areas and sub domains of related work within this single paper. We focus on three important dimensions. We observe that a lot of work in the domain of customizing middleware goes back to the broad base of reflection and to techniques that have conceptually been derived or evolved from this approach. A second area of interest that is essential to the creation of complex middleware that we envisage is the domain of monitoring and observing (without disrupting) distributed software and systems. A third area of importance is the broad area of policy-driven middleware, which we define as middleware that can be described, configured, and managed at a higher abstraction level through the use of policy languages. We believe that these underpinning areas must be revisited and further explored to contribute to the major challenges that we have introduced. In the subsequent subsections, we elaborate on each of the three areas.

Reflection and reflective middleware Reflection has been defined as the capability of a system to reason about itself and act upon this information [26]. For this purpose, a reflective system maintains a representation of itself that is causally connected to the underlying system that it describes [20]. Operations to introspect and make changes to the meta representation are commonly referred to as

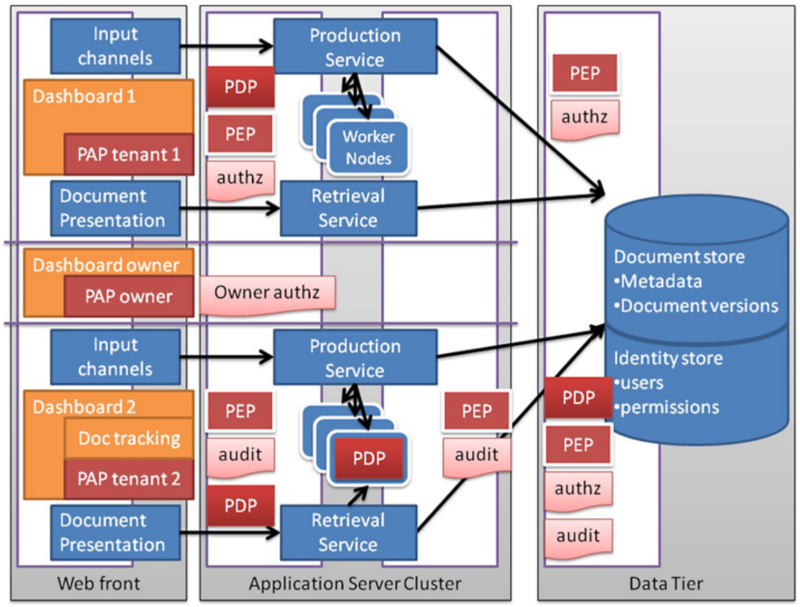

Fig. 3 Variations co-exist at runtime. Multiple stakeholders require different dashboards: the SaaS provider utilizes Dashboard “owner”, the tenants utilize Dashboards 1 and 2, respectively

the Meta Object Protocol (MOP [15]). In component-based frameworks, two styles of reflection have emerged. Structural reflection is concerned with the underlying structure of objects or components and offers support to inspect interface information, and to adapt software compositions. Behavioral reflection is concerned with the activity in the underlying system, e.g. in terms of the sending and dispatching of invocations.

Reflective middleware approaches enable the application and internal middleware structure to be exposed through meta-models [3]. Such meta-models offer *structural reflection* on the interfaces and connections between the application and middleware components. Reflective approaches support extensibility and customization of the middleware through *openness* of the middleware layer. Customizable middleware platforms such as the flexible CORBA platforms (e.g. TAO [17, 24]) increase customizability but typically only at deploy-time. Several reflective middleware platforms, such as OpenCorba [19], DynamicTAO [16], JaCoWeb [30], OpenCOM [8], apply reflective programming to provide runtime customization of the middleware through dynamic operation interception. The Lasagne [29] reflective middleware supports concurrent and co-existent customizations through generic object wrappers.

However, common critiques on reflective middleware state that the runtime overhead is still significant, and that the complexity and impact on pre-existing code often endangers the robustness of the overall (middleware) system. AO-middleware may have brought a partial solution for both problems. AO-middleware uses AOP techniques to compose middleware behavior on well-defined locations in the application [22]. The original work on AOP [14] was inspired by the earlier work on meta object protocols. AOP is often seen as a principled subset of reflective programming. For example, Sullivan [27] first identifies the complex nature of programming reflective systems (*too much rope* for the developer), and secondly states that reflection consumes too much overhead to be a worthwhile technology. He then promotes AOP languages as a means to tame the complexity and reduce overhead. The introduction of richer context information in pointcuts, such as distributed application context in DyMAC [18], has further improved the expressiveness to select well-defined sets of abstractions and operations where customization or monitoring should be enforced. This can further improve the efficiency of policy-based security middleware through selective fine-grained customization or inspection.

Monitoring Monitoring includes two main activities: information extraction and information aggregation. The most common approach to extracting information is *built-in monitoring*. The system is modified by hand to emit events that signal important changes [28]. The disadvantage is that the monitoring system always incurs an overhead, as it is a part of the core system. It cannot evolve independently or be adapted at run-time. As such, this approach is used to expose small volumes of high level information. When the middleware actively supports the emitting of events, such as in Google’s Dapper [25], the event streams of different hosts can be put together to create a distributed trace. The information extracted from such monitoring probes can also be aggregated. For statistical aggregation of monitoring data, collection systems are already widely deployed [10, 31].

A second approach to the extraction of monitoring information is *instrumentation*. Instrumentation systems automatically modify a program so that it emits events. This allows dynamic fine-tuning of the monitoring and its associated overhead. However, dynamic deployment of monitoring probes is a technically complex operation that requires support from the underlying platform. The family of techniques that emerge from the domain of reflective middleware and ao-middleware can deliver, again, an important supporting technology.

Policies in middleware: configuration, deployment, enforcement and management The complexity of current and future middleware platforms demands for higher levels of abstraction to compose, configure, and manage the distributed applications as well as the underpinning middleware and infrastructure (e.g. [9]). This vision has been shared by many, yet most practical examples of new developments and practical solutions reside in the space of public domain software and commercial products. We discuss a couple of relevant examples, and restrict ourselves to the security example that we have presented in Sect. 5.

IBM's Tivoli [11, 13] offers a wide range of management facilities, including some support for the access control needs that we have illustrated in Sect. 2. The enforcement model is however centralized. Tivoli also supports customization of PDPs with custom variants of attribute retrieval. To optimize performance, Tivoli also supports partial policy replication and decision caching. However, many of the needs that have been sketched are not on the radar yet.

Systems for federated identity management have a primary interest in authentication, yet some support for authorization is emerging as well. Shibboleth [12] provides identity attributes (as a PIP), supports caching within a session, but does not take into account attribute freshness when caching. The Globus toolkit [1, 21] supports access control to grid resources (this is comparable to the worker nodes in our case study). However, all attributes in the context store are replicated, limiting its use for sensitive tenant-specific attributes. Globus also supports custom developed PIPs and PDPs. PERMIS [6, 7] also supports a decentralized enforcement model, but with centralized context information stores, which limits performance. It supports customizable PDPs and various PEPs.

All of the technologies provide some customization and configuration of policy enforcement. However, none of these technologies provides cache consistency, attribute freshness, and security guarantees at the same time, and none offer (security) monitoring and management support for multiple stakeholders, i.e. the notion of dashboards. Notice that the presented case study involves both centralized management as well as decentralized management of policies. Centralized management is typically needed for policies defined by the platform owner, policies that obviously must be applied to all tenants. Decentralized management is required for the needs of each individual tenant.

In the industrial context, there are two categories of security middleware that supports policies: the relatively centralized approach (a la Tivoli) and platforms that support decentralized deployment (a la Axiomatics [2, 23]). Tivoli provides centralized management with centralized deployment of policies. Axiomatics provides centralized management but does support decentralized deployment of polices over multiple PDP's. In this research domain, Ponder probably is the most comprehensive management. Ponder also combines centralized policy management with decentralized policy deployment: PEPs are local to the resource and store and enforce the compiled policies. Policy support has been addressed and delivered more or less in isolation from the larger middleware community: solutions are developed and supported as separate tools, without aiming for the cost effective variation and customizations that are typically needed.

7 Bridging the gap

Customization, monitoring, and management of security services in middleware must comply with the architecture of the application. While reflection has focused on implementation-level reification of development-time and runtime concepts, there is an additional need for managing higher-level abstractions while keeping the development cost low and the system performance acceptable. However, the gap between the necessary abstractions to support customization and monitoring on the one hand, and the state-of-the-art support for customization and management on the other hand is excessive. System implementation and deployment would still require too much manual implementation effort.

To manage the development complexity, we need to leverage on model driven techniques to enable automated transformations between abstractions as well as to ensure consistency between development-time and runtime models. To manage performance overhead, we need to leverage on techniques for fine-grained selection and on-demand, just-in-time construction of reflective information.

Cost effective solutions (in man power) could be based on model driven techniques—to map high level to low level abstractions and vice versa—and on complex event processing techniques to gather and correlate run-time events. Ideally, tool support should be based on declarative specifications to (1) select low-level abstractions and events (as, for example in AOP), (2) to map low-level abstractions to more meaningful abstraction (as, for example in model-driven techniques, and (3) to map low-level events to more meaningful events (as for example in the domain of complex event processing). Some initial work shows promising results for distributed systems based on basic RMI middleware [5] and aspect-based middleware [4]. But this seems only the beginning.

8 Summary

Middleware traditionally carries two responsibilities: development support and run-time support for distributed software systems and applications. In this paper, we have discussed two important elements of future middleware needs, and hence future research directions.

Development support must focus on customization (one size fits all will not make it): this is actually not new (this goes back to the roots of reflective middle-ware but we need to be more costs effective—man power wise). Notice that we also need to ensure that the modifications and customizations are compliant with the requirements engineering results and with the software architecture artifacts that have been produced during the entire application engineering process: tenant-specific customizations should not break the guaranteed properties of the main architecture as defined by the platform owner. We believe it is a grand challenge to enable customizations while ensuring this type of integrity in an automated way.

On top of that: we must enable monitoring and management of middleware and the applications it supports. By management, we mean the controlled and informed modifications at run time. This clearly creates a larger mission for middleware vis-a-vis its previous run-time support capabilities. Also, we need richer run time representations of the application and of the corresponding middleware instance while staying cost effective for the various stakeholders. We consider application dashboards to be necessary to deliver internet applications in a trustworthy and manageable way. Yet such a dashboard should be part of the development product of new services and applications. Our paper has set the scene for security services in modern middleware, yet the idea seems applicable to many other needs and concerns.

Acknowledgements This work was partially funded by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, IBBT, the EU-FP7 NoE NESSoS, and by the Research Fund K.U. Leuven. The authors wish to thank Wouter De Borger, Danny Hughes, Sam Michiels, and Stefan Walraven for many fruitful discussions.

References

1. Globus Alliance: Globus Toolkit 4 API. http://www.globus.org/toolkit/docs/4.2/4.2.1/security/
2. Axiomatics: Axiomatics Policy Server 4.0 (2010) http://www.axiomatics.com/products/axiomatics-policy-server.html
3. Blair GS, Coulson G, Robin P, Papathomas M (1998) An architecture for next generation middleware. In: Proceedings of the IFIP international conference on distributed systems platforms and open distributed processing. Springer, London
4. Borger WD, Lagaisse B, Joosen W (2011) Traceability between run-time and development time abstractions. In: Jane Cleland-Huang OG, Zisman A (eds) Software and systems traceability. Springer, Berlin
5. Borger WD, Lagaisse B, Joosen W (2011) A generic solution for agile run-time inspection middleware. In: Middleware'11.
6. Chadwick D, Zhao G, Otenko S, Laborde R, Su L, Nguyen TA (2008) Permis: a modular authorization infrastructure. Concurr Comput Pract Exp 20:1341–1357
7. Chadwick DW, Su L, Laborde R (2008) Coordinating access control in grid services. Concurr Comput Pract Exp 20:1071–1094
8. Clarke M, Blair G, Coulson G, Parlavantzas N (2001) An efficient component model for the construction of adaptive middleware. In: Middleware 2001, pp 160–178
9. Delaet T, Joosen W (2007) Podim: a language for high-level configuration management. In: Proceedings of the 21st conference on large installation system administration conference. USENIX association
10. Delgado N, Gates AQ, Roach S (2004) A taxonomy and catalog of runtime software-fault monitoring tools. IEEE Trans Softw Eng 30(12):859–872
11. IBM: IBM Tivoli Access Manager. http://www-01.ibm.com/software/tivoli/products/access-mgr-e-bus/
12. Internet2MiddlewareInitiative/MACE: Shibboleth 2. http://wiki.shibboleth.net/confluence/display/SHIB2
13. Karjoth G (2003) Access control with ibm tivoli access manager. ACM Trans Inf Syst Secur 6(2):232–257
14. Kiczales G, Lamping J, Menhdhekar A, Maeda C, Lopes C, Loingtier JM, Irwin J (1997) Aspect-oriented programming. In: Proceedings European conference on object-oriented programming, vol 1241. Springer, Berlin
15. Kiczales G, Rivières JD, Bobrow DG (1991) The art of the metaobject protocol. MIT Press, Cambridge
16. Kon F, Román M, Liu P, Mao J, Yamane T, Magalhães C, Campbell RH (2000) Monitoring, security, and dynamic configuration with the dynamicTAO reflective ORB. In: Middleware'00: IFIP/ACM international conference on distributed systems platforms. Springer, New York
17. Kuhns F, O'Ryan C, Schmidt D, Othman O, Parsons J (1999) The design and performance of a pluggable protocols framework for object request broker middleware. In: Proceedings of the IFIP, vol 6
18. Lagaisse B, Joosen W (2006) True and transparent distributed composition of aspect-components. In: Middleware'06: proceedings of the ACM/IFIP/USENIX 2006 international conference on middleware. Springer, New York
19. Ledoux T (1999) OpenCorba: a reflective open broker. In: Reflection'99. Springer, London
20. Maes P (1987) Concepts and experiments in computational reflection. In: OOPSLA'87. ACM, New York
21. Malhotra D (2011) Devanand: Mgc middleware for grid computing: the globus toolkit. In: Proceedings of the international conference on advances in computing and artificial intelligence, ACAI'11. ACM, New York
22. Pawlak R, Duchien L, Florin G, Seinturier L (2001) Jac: A flexible solution for aspect-oriented programming in java. In: Metalevel architectures and separation of crosscutting concerns
23. Rissanen E, Brossard D, Slabbert A (2009) Distributed access control management—a xacml-based approach. In: ICSOC-servicewave. Springer, Berlin
24. Schmidt DC, Levine DL, Mungee S (1998) The design of the TAO real-time object request broker. Comput Commun 21(4):294–324
25. Sigelman BH, Barroso LA, Burrows M, Stephenson P, Plakal M, Beaver D, Jaspan S, Shanbhag C (2010) Dapper, a large-scale distributed systems tracing infrastructure. In: Google research
26. Smith BC (1982) Reflection and semantics in a procedural language. Ph.D. thesis, MIT
27. Sullivan GT (2001) Aspect-oriented programming using reflection and metaobject protocols. Commun ACM 44(10):95–97

28. Sun Microsystems I.: Java management extensions (2009) http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/
29. Truyen E, Vanhaute B, Joosen W, Verbaeten P, Jorgensen BN (2001) Dynamic and selective combination of extensions in component-based applications. In: Proceedings of the 23rd international conference on software engineering, ICSE 2001
30. Wangham MS, Lung LC, Westphall CM, Fraga JS (2001) Integrating SSL to the JaCoWeb security framework: project and implementation. In: Proceedings of the 7th international symposium on integrated network management–IM
31. Zanikolas S, Sakellariou R (2005) A taxonomy of grid monitoring systems. Future Gener Comput Syst 21:163–188