JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2004**

Vol. 3, No. 2
Special issue: .NET: The Programmer's Perspective: ECOOP Workshop 2003

Support for Metadata-driven Selection of Run-time Services in .NET is Promising but Immature

Frank Piessens, **Bart Jacobs**, **Eddy Truyen**, and **Wouter Joosen**
Department of Computer Science, K.U.Leuven, Belgium

Abstract

The .NET Framework allows developers to add run-time services to their classes by specifying them in metadata. This metadata-driven service selection is a very powerful and promising mechanism, closely related to ideas developed in the Aspect-Oriented Programming community. Interestingly, the .NET framework supports both services implemented by weaving and services implemented by interception. However, the weaving-based and the interception-based mechanisms seem to have been introduced in the framework independently, and show some unnecessary differences in flexibility, extensibility and configurability. Also both mechanisms still contain some anomalies in their design. In this paper, we describe the mechanisms, and discuss these shortcomings.

1 INTRODUCTION

The Microsoft .NET Framework improves on existing component systems, such as Java and J2EE, COM and COM+, and CORBA, in a number of ways. It combines many good features of each of these predecessors, while also including some innovations, such as:

- Components are named using precise yet friendly names. Names include version and culture, which provides bind-time flexibility. Namespace ownership is cryptographically enforced.
- Arbitrary metadata can be added to components and elements within components, for use at compile-time (both source compile-time and JIT compile-time), at link-time, or at run-time. At source compile-time, metadata labeling methods with the exceptions they can throw can be used by the compiler to perform similar checks as a Java compiler does for checked exceptions. Some types of metadata trigger the execution of services directly by the framework and the service code (call) is

Cite this article as follows: Frank Piessens et al.: "Support for Metadata-driven Selection of Run-time Services in .NET is Promising but Immature", in *Journal of Object Technology*, vol. 3, no. 2, Special issue: .NET: The Programmer's Perspective: ECOOP Workshop 2003, pp. 27-35.
http://ww.jot.fm/issues/issue_2004_02/article3

inserted at JIT compile-time. Other types of metadata trigger link-time checks, for instance a security check whether a component is permitted to link to a given method. Most types of metadata can be retrieved at run time by user code or by the framework for various purposes, such as determining the serializability of a class.

- Better support for independent extensibility: when a developer introduces a late-bound method in a class, they need to explicitly state whether they intend to override a base class method or not. If a method is later added to the base class, it cannot inadvertently be overridden by an existing method in the derived class.
- From a security perspective, making highly privileged components globally available on a machine does not automatically imply an increased attack surface, since partially trusted clients only have access to those components that have explicitly been marked as safe for such access by the developer.

All these features together provide for much improved support for component-based development [Szyperski 2002].

In this paper, we focus on how the .NET framework uses metadata to support the activation of various run-time services, such as access control, synchronization, transactions and so forth. The .NET framework is the first commercial product to provide an extensible system for defining metadata-driven run-time services. The authors feel that this approach is very promising, but the current implementation in .NET still has some conceptual and technical shortcomings.

2 METADATA-DRIVEN SELECTION OF RUN-TIME SERVICES

Developers add metadata to types, members and other elements by annotating them with so-called attribute specifications. An attribute specification consists of a type name which names an attribute class, plus an argument list consisting of literal expressions. The type name and the literal values are stored in the assembly by the programming language compiler.

At run-time, this information is used by the CLR to create an instance of the named attribute class. The CLR and the application itself can retrieve the instances associated with an element, look for specific types of instances, and act upon them.

A fairly large number of attribute classes are predefined in the .NET class library. These predefined attributes are used in a variety of ways in the CLR, e.g. to indicate if classes are serializable, or to drive optimizations. One particular use of attributes that concerns us in this paper is the triggering of run-time services in the CLR. Developers can decorate their components with such attributes to indicate to the runtime that certain cross-cutting services (such as access control or synchronization) should be activated at run-time.

We discuss two examples of attributes that request run-time services: security attributes and context attributes.

Security attributes

Security attributes derive from `SecurityAttribute`; they instruct the CLR to perform named security actions when the element is accessed [LaMacchia 2002]. For example:

```
public class Robot {
    [RobotPermissionAttribute(
         SecurityAction.Demand, RobotAction.GetLocation)]
    RobotLocation GetLocation() {
        // ...
    }

    [RobotPermissionAttribute(
         SecurityAction.Demand, RobotAction.MoveTo)]
    void MoveTo(RobotLocation location) {
        // ...
    }
}
```

For security attributes, the CLR provides the required service by adding extra code when a method is compiled to native code, i.e. essentially by a simple form of run-time code weaving. The added code performs the requested security actions. Code weaving can be applied to static methods as well as instance methods and is a fairly efficient mechanism.

Context attributes

Context attributes [Lowy 2003] are attribute classes that derive from `ContextAttribute`. They should be applied to a class which derives from `ContextBoundObject`. Context-bound objects live in a so-called context, that is determined by a number of context properties. These properties essentially say what run-time services the context provides. On instantiation of a context-bound object, the runtime inspects the metadata, and decides in which context the object should be instantiated. If no suitable context is available, a new context providing exactly the required services is created.

Messages to context-bound objects sent from outside the object's context are intercepted and reified, and can then be manipulated by a number of message sinks. These sinks can do pre-processing, post-processing, collect state, manipulate the message and so forth, to provide a certain run-time service.

The mechanism of contexts is extensible by developers: new context attributes can be defined. Such a newly defined attribute can contribute new properties to a context, and can add a new message sink to the context-bound object's interception chain.

The .NET Framework currently includes a single predefined context attribute class, called SynchronizationAttribute. However, it is expected that some or all of the run-time services that are now provided through COM+ will be made available in the form of context attributes in a future version of the framework.

An example of the use of a context attribute, SynchronizationAttribute, is as follows:

```
[SynchronizationAttribute]
public class Account : ContextBoundObject {
    private int balance;

    public void deposit(int amount) {
        balance += amount;
    }
}
```

Essentially, the mechanism of context-bound objects is a generalization of the idea of container-provided cross-cutting services, as was present already (in a non-extensible way) in COM+, J2EE and CORBA.

This interception-based approach of adding run-time services is more expressive, but less efficient compared to the weaving-based approach.

3 DISCUSSION

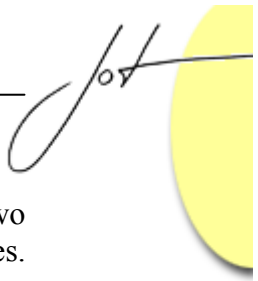
The idea of using metadata to drive activation of run-time services is very promising (and in fact has been suggested in the research community before the release of .NET). Metadata-driven selection of services can be considered to be a kind of aspect-oriented programming [Kiczales 1997, Shukla 2003]: the pointcut is determined by the metadata on the static program elements (components, classes and methods), whereas the advice is either the code that is woven in, or the code that is executed by the message sinks.

The current implementation in the CLR however, has some drawbacks. In this section, we will discuss these drawbacks and suggest some improvements.

Ad-hoc and separated definitions of the two mechanisms

It is clear that certain run-time services could be supported both by weaving and by interception. In the current CLR for example, the PrincipalPermissionAttribute can be used to do role-based access control implemented by weaving, and the SecurityRoleAttribute can be used to do role-based access control implemented by interception.

However, it seems as if both mechanisms have found their way into the CLR independently: the weaving-resolved attributes seem to have been introduced to deal with mobile code security and the context attributes are a generalization of contextual



composition in COM+. As a consequence, the conceptual similarities between the two approaches are not brought out clearly at the level of, for instance, the attribute classes. This is unfortunate. We see two opportunities for improvement:

- Conceptually similar design for both mechanisms.
  Since weaving-based and interception-based implementations each have their own advantages and disadvantages, it is clear that both techniques should remain present in the CLR. However, by more clearly showing the conceptual similarity, one could make it easier to switch between the two types of implementation for a given run-time service.
  Since the choice between weaving and interception is often a trade-off between flexibility and efficiency, such switching could be used in optimizations.
- Flexible weaving mechanism.
  The interception-based mechanism has been designed to be developer-extensible. Developers can define new context attributes that specify new message sinks to be incorporated in the interception chain.
  The weaving-based mechanism is not extensible at all: the CLR recognizes a fixed number of predefined attributes, and there is no way for developers to define new such attributes and specify what code should be woven.
  It would be interesting, certainly from a research point of view, to have a runtime where both mechanisms are extensible.

Limited expressive power

The CLR only supports the addition of attributes to the declaration of static program structures (assemblies, classes, methods, ...), and not to dynamic structures (e.g. interactions, objects). As a consequence, what services are provided for a specific object is fixed at instantiation time of the object, and no client-specific customizations are supported.

The CLR does contain the necessary low-level concepts to support such client-specific customizations: the LogicalCallContext class is a collection object that carries information about the current logical thread of execution, even across remoting calls. It could be used to carry metadata about the current interaction.

The Lasagne customization model [Truyen 2001] takes this idea to its extreme. Instead of allowing attribute specifications scattered across the program code, Lasagne proposes to externalize all such specifications in a separate first class entity, called a composition policy object, that automatically propagates with the logical control flow of subsequent interactions. As such, service selection logic travels as metadata with the locus of execution, rather then being locked up and scattered across the code of the program. In this approach programmer-driven selection of services can still be supported since the composition policy object can be inspected and manipulated at any execution point.

Example

The delivery of non-repudiation evidence of the result of a method call can be programmed as a server-side message sink (that signs incoming parameters, result, and any other information as required, for example). A client-side message sink can then verify this and save it for later use.

For this example, it is definitely useful that the client chooses whether this service is activated for a specific method call.

Unclear composition model for run-time services

While the CLR supports extensible interception chains providing an arbitrary number of services, it is not clear how to deal with services that are not completely orthogonal. For such services, developers should be able to specify at least the ordering of the interceptors, and possibly also interfaces between dependent services. There is no support for this available in the framework.

Adding support to specify dependencies between services to determine the ordering of message sinks, and enabling some form of communication between dependent sinks is relatively straightforward in the current model. Also, some limited support to detect conflicting services is built in: after a new context has been created, each service is queried whether the resulting context is ok or not. If one of the installed services answers negatively, an exception is thrown.

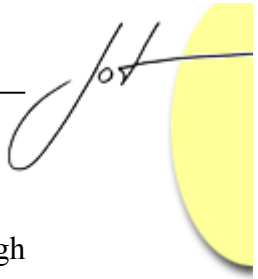
However, since general composition of aspects is itself still an active research topic, there are no simple general solutions to be expected.

The issues surrounding composition of services are discussed by Szyperski in [Szyperski 2002]. He gives the example of composing a logging service and an encryption service; he notes that the effect of the composition depends on the order in which these services are composed, and that for some applications a cooperation between these services is required that is more specialized than a simple execution of one service after the other.

Limited support for configuration of run-time services by administrators

Selecting run-time services by decorating static program structures with attributes provides developers with a powerful mechanism to choose appropriate services. However, for some services (for instance, access control), the deployer and administrator of an application should also have their say, for instance by means of a configuration mechanism. For COM+ services, both metadata-driven and configuration data-driven selection of services is possible already. For context-bound objects in the CLR, support for configuration of services is possible by means of dynamic context properties (but the current CLR does not yet seem to provide tool support for this).

For the weaving-resolved attributes however, only development-time metadata is taken into account. As a consequence, if the developer did not add any security attributes



to the code, there is currently no easy way for a deployer/administrator to add them.
Of course, enabling administrators to add arbitrary run-time services through configuration also carries some risks: in an extensible system, services can add arbitrary code and hence the semantics of a component could change radically through reconfiguration. This is clearly undesirable.

A possible solution could force developers of run-time services (e.g. developers of context attributes) to indicate if the service could be selected through metadata, through configuration or both.

Low-level programming model

Programming services as message sinks is not the most developer-friendly programming model. Moreover, there is little type-checking on composition of sinks. Programming a service as a decorator of a class solves these two problems, and such a decorator could be compiled to a message sink. However, in such a model, services are less polymorphic in the sense that they cannot easily be applied to many classes. The challenge here is to design a programming model that combines the ease-of-use and type-safety of decorators, with the polymorhphy of message sinks.

For example, the CAESAR [Mezini 2003] programming model is a step towards resolving this challenge. It proposes the notion of collaboration interface [Mezini 2002] as a higher-level module concept on top of aspect-oriented join point interception. A collaboration interface allows to separate the implementation of an aspect - specified in an aspect implementation, from how to connect that implementation with a particular application, which is specified in an aspect binding. Aspect implementation and aspect binding are indirectly connected to each other since they implement two loosely coupled facets of the collaboration interface. This loose coupling is the key to polymorphic reuse of implementations and bindings. Since collaboration interfaces are typed, the type-safety of decorators is provided while the loose coupling between implementation and binding provides the polymorphy of message sinks. An interesting application of this programming model in the context of metadata-driven selection of services is to split message sinks as well in an implementation part and a binding part and have the binding part automatically be generated from the programmer-specified metadata.

4 CONCLUSION

The support for metadata-driven service selection is powerful and promising. In the research community, it has long been recognized that weaving and interception are two powerful techniques to compose cross-cutting services with an application. The .NET framework supports both techniques, but in a very asymmetric way: the weaving-based approach is tuned to support only a small number of built-in services, whereas the interception-based approach is designed to be easily extensible. Also, both approaches still allow for improvements in configurability, usability, and composability of provided services.

REFERENCES

[Kiczales97] G. Kiczales, J. Irwin, J. Lamping, J.-M. Loingtier, C. V. Lopes, C. Maeda, and A. Menhdhekar. "Aspect-oriented programming". In M. Akşit and S. Matsuoka, editors, *ECOOP '97 — Object-Oriented Programming 11th European Conference, Jyväskylä, Finland*, volume 1241, pages 220–242. Springer-Verlag, New York, NY, 1997.

[LaMacchia02] B. LaMacchia, S. Lange, M. Lyons, R. Martin, and K. Price. *.NET Framework Security.* Addison Wesley, 2002.

[Lowy03] J. Lowy. "Contexts in .NET: Decouple components by injecting custom services into your object's interception chain". *MSDN Magazine*, March, 2003.

[Mezini02] M. Mezini and K. Ostermann. "Integrating independent components with on-demand remodularization". In *Proceedings of the 17th ACM conference on Object-oriented programming, systems, languages, and applications (OOPSLA-02)*, volume 37(11) of *ACM SIGPLAN Notices*, pages 52–67. ACM Press, Nov. 4–8 2002.

[Mezini03] M. Mezini and K. Ostermann. "Conquering aspects with CAESAR". In *Proceedings of the 2nd International Conference on Aspect-Oriented Software Development (AOSD)*, pages 90–100. ACM Press, March 17-21 2003.

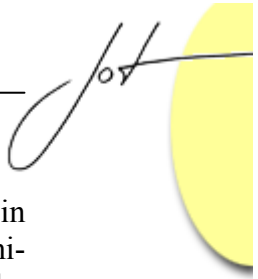
[Shukla03] D. Shukla, S. Fell, and C. Sells. "Aspect-oriented programming enables better code encapsulation and reuse". *MSDN Magazine*, March, 2003.

[Szyperski02] C. Szyperski, D. Gruntz, and S. Murer. *Component Software: Beyond Object-Oriented Programming, 2nd Ed.* Pearson Education, 2002.

[Truyen01] E. Truyen, B. Vanhaute, W. Joosen, P. Verbaeten, and B. N. Jørgensen. "Dynamic and selective combination of extensions in component-based applications". In *Proceedings of the 23rd International Conference on Software Engineering (ICSE'01)*, pages 233–242. IEEE Computer Society, May 12–19 2001.

About the authors

Frank Piessens is a professor at the Department of Computer Science of the Katholieke Universiteit Leuven in Belgium. His research interests are in security aspects of software, including security in middleware and operating systems, security architectures, formal methods for security, Java and .NET security, and software interfaces to security technology. He can be reached at frank.piessens@cs.kuleuven.ac.be.



Bart Jacobs is a PhD student at the same Department. His research interests are in formal methods for security, including type systems, calculi, automated and semi-automated reasoning, programming language semantics, and secure platforms. He can be reached at bart.jacobs@cs.kuleuven.ac.be.

Eddy Truyen is a PhD student at the same Department. His research interests are in language technology and middleware, with a focus on client-specific and dynamic customization of distributed systems. He can be reached at eddy.truyen@cs.kuleuven.ac.be.

Wouter Joosen is a professor at the same Department. He is a co-founder and director of Ubizen N.V. and Luciad N.V., two spin-offs of the Department. Wouter's research interests are distributed systems and network applications, with a focus on software security, agent-based systems, real-time systems and development environments. Wouter Joosen is also an Adjunct Professor for Software Engineering at the University of Odense, Denmark. He can be reached at wouter.joosen@cs.kuleuven.ac.be.