Dynamic Hook Points

Eli Acherkan[1,2], Atzmon Hen-Tov[2], David H. Lorenz[1],

Lior Schachter[1], Rebecca Wirfs-Brock[3], Joseph W. Yoder[4]

[1]The Open University of Israel, Raanana 43107, Israel

[2]Pontis Ltd., Glil Yam 46905, Israel

[3]Wirfs-Brock Associates

[4]The Refactory, Inc.

eliac@pontis.com, atzmon@pontis.com,
lorenz@openu.ac.il, liorsav@gmail.com,
rebecca@wirfs-brock.com, joe@refactory.com

Abstract. *When building dynamic systems, it is often the case that new behavior is needed which is not supported by the core architecture. One way to vary the behavior quickly is to provide well-defined variation points, called hook-points, at different places in the systems, and have a means to dynamically lookup and invoke new behavior at runtime when desired.*

Categories and Subject Descriptors

D.1.5 [**Programming Techniques**]: Object-oriented Programming; D.2.2 **[Design Tools and Techniques**]: Object-oriented design methods; D.2.11 **[Software Architectures]**: Patterns

General Terms

Design

Keywords

Adaptive Object-Models, Dynamic systems, Reflection, Variation points, Hook points

Introduction

It is often necessary to adapt the behavior of an Adaptive Object-Model (AOM) system [YBJ01; YJ02] in a way not supported by the core AOM architecture. One way to extend the behavior quickly and without going through a complete build-redeploy software delivery cycle is to provide well-defined hook-points at different places in the domain model where custom behavior can be defined.

It is also highly desirable to enable custom behavior to be defined by the AOM user, who isn't necessarily a programmer. One way to achieve this is by providing the AOM user with the ability to define Business Rules [Ars00]; that is, to specify actions to be taken when certain conditions are met.

Sometimes, complex custom behavior must interact with external systems or data repositories. In such cases, writing imperative code cannot be avoided. Often using a lightweight scripting language is a good way to add custom code. Sometimes, however, the extra requirements of a particular custom behavior demands highly optimized performance, which requires programming in the hosting language (e.g., Java, C#) to implement the hook.

Intent

This DYNAMIC HOOK POINTS pattern is intended primarily for those that are building dynamic or flexible AOM systems and need ways to incorporate variation points in the architecture where new behavior can be added dynamically without a full compile-build-deploy cycle.

Context

In an AOM, the model is often revised as new business requirements emerge. These modifications to the AOM may also require changes in the system behavior not anticipated by the AOM developer. While the proper way to address such changes is to modify the AOM domain entities, this entails new software delivery and redeployment of the system.

Problem

How can you allow a system that is changing in specific locations to adapt to unpredicted behaviors without deploying a new software version? These changes are in well-known places, but the changes are not known in advance.

Forces

- ***Customizability***: There are many customers using the product, each customer may modify their own AOM model.
- ***Adaptability***: The AOM model changes over time. The variations are required to withstand model changes.
- ***Extensibility***: Over time, new locations in the system may be recognized as apt to change. The development effort required to introduce a variation point in a new location should be taken into account.

- ***Crosscutting concerns***: The varying behavior should be subject to the same common design practices as enforced in the rest of the system, such as logging, performance tracing, persistence validation (e.g., type safety), and security auditing.
- ***Scoping***: The variations need a well-defined scope, which controls what they can and cannot do, and which parts of the system are accessible to them and can be affected by them.
- ***Reuse***: The same behavior may be required in several variation points. On the one hand, a solution that promotes reuse (e.g., via libraries or inheritance) is preferable to a solution that forces duplication of behavior. On the other hand, supporting reuse may introduce more complex regression problems, e.g., when changing a shared component all dependent components should be verified for consistency, preferably using static analysis.
- ***Testability***: The solution should support users in testing the behavior extension before launching it to production.

Solution

Analyze your business flows and identify the places that should support behavioral variability. Modify the behavior of the `Entities` in those well-defined places to invoke dynamic hooks. Employ AOM mechanisms to connect dynamic hook implementations to the user-defined AOM classes, by letting your `EntityTypes` hold dynamic hooks as data members, thus connecting a dynamic hook instance to a user-defined class. The class diagram in Figure 1 depicts the main classes involved in the definition and invocation of dynamic hooks.

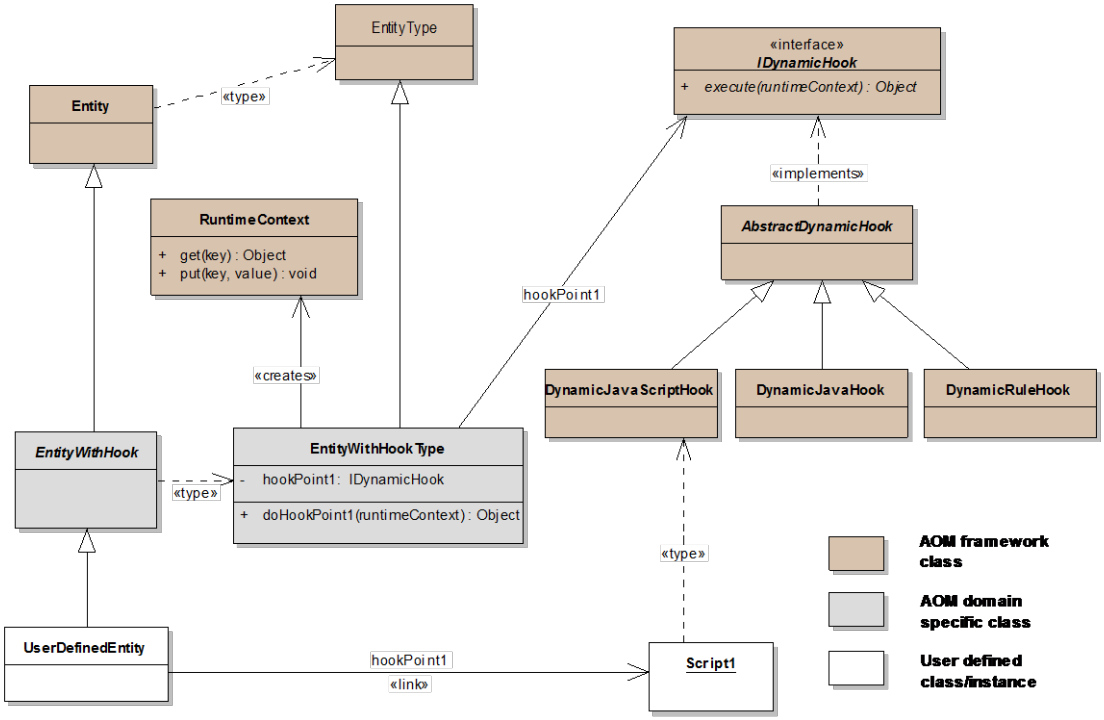

Figure 1 - Class diagram of the main classes involved in hook points

`IDynamicHook` represents a generic interface containing a single `execute()` method. The interface is implemented by the abstract base class `AbstractDynamicHook`. `JavaDynamicHook`, `JavaScriptDynamicHook` and `RuleDynamicHook` are different types of hooks that allow users to extend the system behavior in Java, JavaScript and Rules respectively.

`EntityWithHook` is an AOM core application entity that is to be subclassed dynamically via AOM. It allows a dynamic hook to adapt the behavior of its user-defined subclasses (e.g., `UserDefinedEntity`). Its EntityType is `EntityWithHookType`.

`EntityWithHookType` contains a property `hookPoint1` of type `IDynamicHook`. When the AOM user defines a subclass of `EntityWithHook` (and therefore an "instance" of `EntityWithHookType`), the user can supply an instance of `IDynamicHook` as the value for the `hookPoint1` property. The dynamic hook instance can be of any concrete subclass of `IDynamicHook` (e.g., `Script1` is an instance of `DynamicJavaScriptHook` in Figure 1).

The dynamic hook is invoked by `EntityWithHook` in the appropriate place in the flow. The data that `EntityWithHook` passes to the dynamic hook is encapsulated in an instance of `RuntimeContext`. In its simplest form, it can be a map of key-value pairs. A `RuntimeContext` is created and populated by `EntityWithHook` prior to each invocation.

Using a common interface (`IDynamicHook`), the caller (`EntityWithHook`) is unaware of the concrete type of the dynamic hook supplied (e.g., Java, JavaScript, or Rule-based). Moreover, the same property (`hookPoint1`) can contain different types of hooks in different subclasses of `EntityWithHook`.

A single Entity can have several hook points and invoke several dynamic hooks during its execution. Some of the hook points can invoke the same dynamic hooks, while other hook points invoke different hooks (for example, `EntityWithHookType` can have data members `hookPoint1` and `hookPoint2`, and `EntityWithHook` may have two points in the logic flow where it invokes `hookPoint1`, and only one from which it invokes `hookPoint2`).

Figure 2 shows the sequence of operations during dynamic hook invocation.

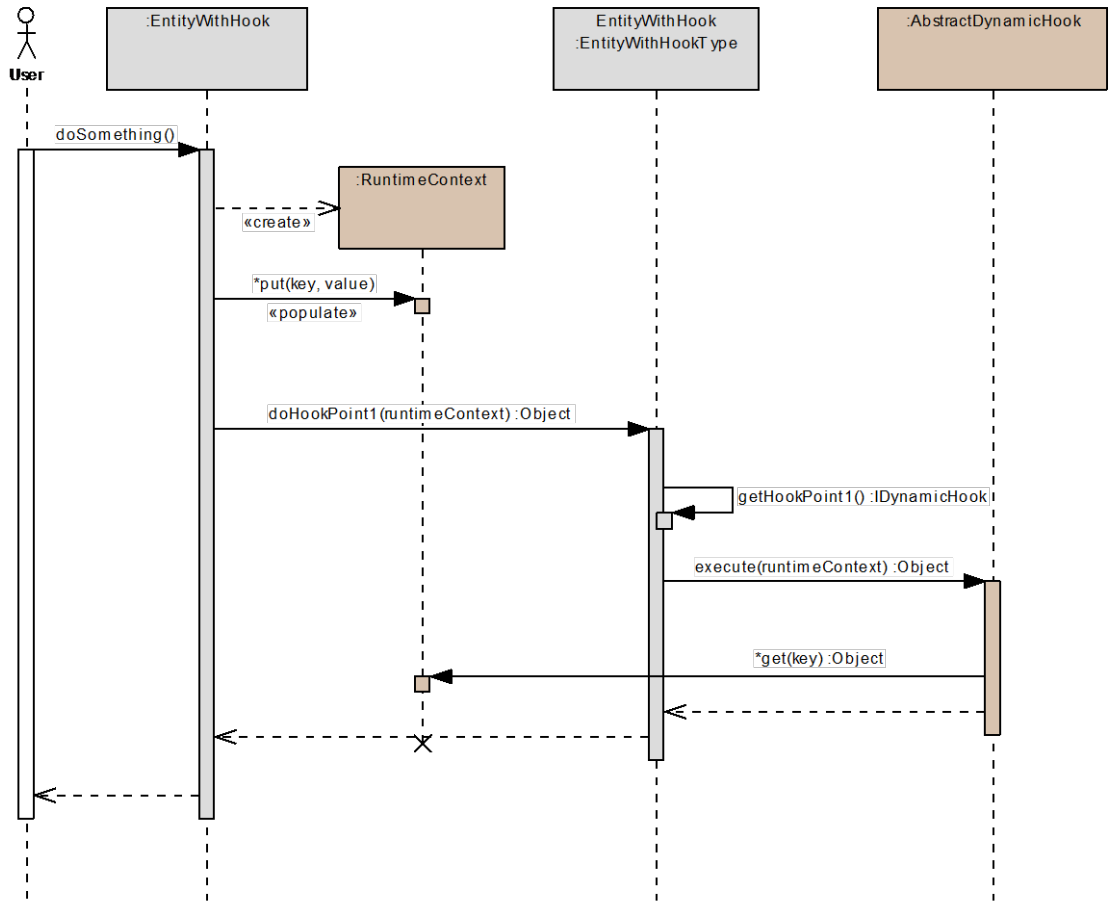

Figure 2 – Sequence diagram of dynamic hook invocation

A client invokes an operation (`doSomething`) on an instance of `UserDefinedEntity`. `UserDefinedEntity` is a dynamic subclass of `EntityWithHook`.

At the appropriate place in its execution of `doSomething`, `EntityWithHook` invokes the corresponding dynamic hook by creating an instance of `RuntimeContext`, populating it with the data that should be accessible to the hook's body, and calling the `doHookPoint1` method of `EntityWithHookType`.

`EntityWithHookType` retrieves the value of its `hookPoint1` property, and (if the value exists) invokes the `execute` method. The dynamic hook performs its logic (accessing the `RuntimeContext` if needed), optionally returning a value. When control returns to `EntityWithHook`'s `doSomething` method, it processes the returned value if needed, and proceeds.

Example

Comfortable Couch (CC) is a fictitious conference management system. CC supports submissions of papers, assignment of submissions to reviewers, and audits the

submissions lifecycle. In CC one can customize the system for each conference. For example, the conference chairs can customize the review form.

One of the conference chairs requested from the CC product management a new feature: every time a reviewer submits a review, the conference chair wants to be notified with a text message (SMS). Since sending text messages requires integration with many service providers and there are charges per message transmission, the CC product-board is reluctant to develop this capability as part of the product. However, they remembered a few old requests made by the conference chairs for adapting the behavior of the system upon a submission of a review. A researcher once asked for the ability to track the "time-before-deadline" of review submissions as part of a research on "the student syndrome among PC members." Another chair wanted to validate the review form and to deny a submission in which a poor rating is not accompanied by a well-justified explanation of sufficient length (e.g., a rule that states that if the technical quality of a paper is graded under 3, then the "technical quality comments" field must contain at least 200 characters).

After considering the varying needs for behavior adaptation in the review form's lifecycle, the CC product-board decided to open a new hook point to allow each chair to customize the behavior for his conference.

Luckily, Comfortable Couch is developed using the AOM architecture style, which provides for a natural solution (object-oriented wise) for fitting the hook into the design.

Figure 3 shows the main classes involved in the use-case.

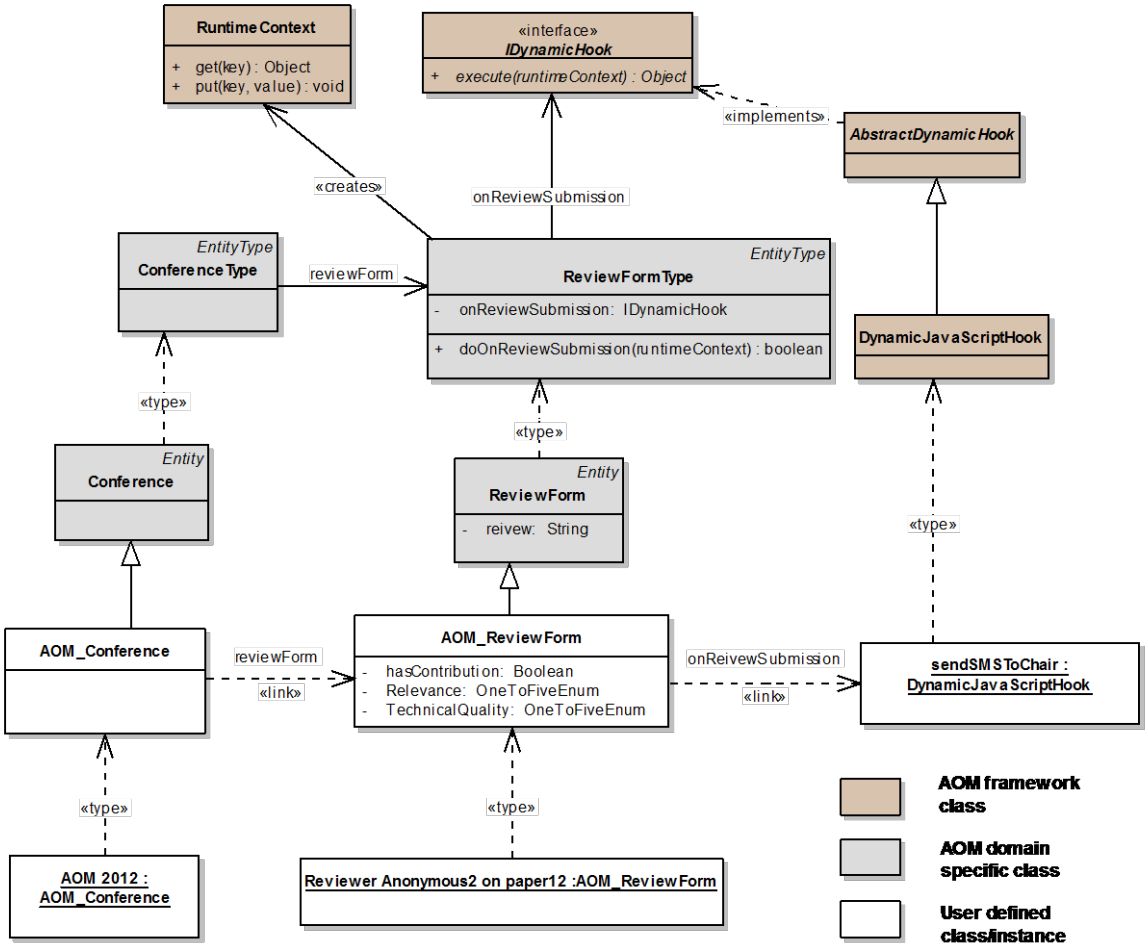

Figure 3 - Example class diagram

`Conference` and `ReviewForm` are AOM Classes. A `Conference` specification includes the type of review form to be used (`<<link>>` from `AOM_Conference` to `AOM_ReviewForm` in Figure 3). Defining the new hook point in `ReviewForm` resulted in adding the data member `onReviewSumbission` and the method `doOnReviewSubmission` to the EntityType `ReviewFormType`. This new structure allows the conference chair (AOM user) to customize the behavior of CC by specifying an instance of `IDynamicHook`, such as `sendSMSToChair` in Figure 3.

The `ReviewForm`'s business logic is implemented in the `ReviewForm` AOM application class. In the proper point in the logic flow the hook is invoked. This is done by populating the `RuntimeContext` (possibly with objects such as the review form, reviewer and paper) and invoking the `doOnReviewSubmission()` method on the `ReviewFormType`, the implementation of which delegates the processing to the `onReviewSubmission` data member. In Java it may look like this:

```
boolean doOnReviewSubmission(RuntimeContext runtimeContext) {
   Boolean result = true;
   IDynamicHook onReviewSubmissionHook = getOnReviewSubmission();
   if (onReviewSubmissionHook != null) {
      result = (Boolean)onReviewSubmissionHook.execute(runtimeContext);
   }
```

```
    return result;
}
```

The `ReviewForm` allows the dynamic hook to stop the submission process by returning `false`:

```
RuntimeContext context = prepareContext();
boolean shouldProceed = getEntityType().doOnReviewSubmission(context);
if (shouldProceed) {
   // ....
}
```

Implementation notes

- Adding a *list* of dynamic hooks as a data member to an EntityType (instead of a single object) allows the user to define several behavior extensions that will operate sequentially, aiding separation of concerns.
- There are several techniques for invoking a scripting language from a separate host language. For example, the Java Development Kit (JDK) includes a scripting engine with built-in support for JavaScript.
- In Java, hooks can be implemented either as a plugin in an OSGi environment e.g., Equinox [EEQX] or with on-the-fly Java compilation (using the Java Compiler API in a JSP-like manner).
- Rules can be implemented with expression-languages. For example, in Java using Spring EL [SEL], JSF EL [JSFEL] and JXPath [JXP].
- The ability to reuse a dynamic hook depends on the implementation. Possible solutions include allowing many-to-one references to instances of `IDynamicHook`, or multi-level inheritance between user-defined AOM classes.

Consequences

✓ ***Time to market:*** Changes to the architecture or new behaviors can be added without the need for another complete compile-build-deploy cycle.

✓ ***Increased flexibility:*** It is possible to choose on a per-case basis whether to use a scripting language, your favorite programming language, or a RULE OBJECT pattern language [Ars00]. If the variation needed is of limited expressiveness then using rules is preferable, since it provides a controlled solution with good performance (rules are implemented in the system programming language). If expressiveness is a major concern, then a scripting language can be used. When both expressiveness and performance are critical, hooks can be implemented using the system programming language and introduced to the system using reflection or other methods.

✓ ***Ease of use:*** Definition of a new hook point (performed by the application team) and implementation of a dynamic hook (performed by the AOM user) require a relatively small effort.

× ***Higher complexity:*** The usage of dynamic hooks increases complexity through the addition of a new level of indirection and interpretation when binding the hook context to the dynamic hook implementation. This extra level of complexity also requires more sophistication in debugging and testing. Testing of the system can be

more complex, because tests must be provided not only for the core AOM architecture, but also for all the hook points to make sure they work properly in conjunction with the core AOM system.

- ***Performance overhead:*** Using scripting languages to implement hooks can impact performance. Using caching mechanisms can eliminate the performance overhead related to just-in-time compilation and creation of temporary objects.
- ***Increased cohesion:*** The dynamic hooks must follow the contract defined by the AOM application layer. This increases the number of dependencies between components, and puts the dynamic hooks at risk when the AOM application is changed. This is particularly important for hooks that don't support type safety. The EVOLUTION RESILIENT SCRIPT pattern [HNS10] addresses this issue and improves type safety.

Related Patterns

The STRATEGY pattern describes a similar mechanism for non-AOM systems, where both the hook point and its implementation reside in application classes. The STRATEGY pattern doesn't address dynamic modification of the behavior at runtime.

The EVOLUTION RESILIENT SCRIPT pattern enhances DYNAMIC HOOKS by providing type safety and ongoing validation.

Known Uses

Pontis Ltd. (www.pontis.com) is a provider of Online Marketing solutions for Communication Service Providers. Pontis' Marketing Delivery Platform (MDP) allows for on-site customization and model evolution by non-programmers. The system is developed using ModelTalk [HLPS09] based on AOM patterns. Pontis' MDP system is deployed in over 20 customer sites including Tier I Telcos. A typical customer system handles tens of millions of transactions a day exhibiting Telco-Grade performance and robustness.

Pontis' MDP system aggregates data received from the Communication Service Provider's systems, such as information about a subscriber's usage patterns, and grants various benefits to subscribers based on the subscriber's data and the currently active promotions (e.g., a subscriber that sent 100 text messages receives a promotional coupon).

The DYNAMIC HOOK POINTS pattern is widely used in MDP. One such hook point is invoked whenever a benefit is granted to a subscriber. Each customer project and each benefit class (represented as a user-defined class) can augment the system's behavior by writing code (in either JavaScript or Java) that will be invoked every time a benefit is granted. Several projects use this hook point in order to record the details of certain classes of benefits into an external data warehouse system.

Two adaptive systems for Invoicing and Import developed by The Refactory (www.TheRefactory.com) in C#/.NET used dynamic hook points to define known places to add new behavior. One dynamic hook point in the Import system was for adding new rules. New rules can be added by creating a DLL, which contains a subclass of `ValidationRule`. This class will be tagged with the name of the validation rule and have a `Validate` method which is invoked during the validation process. By

including the DLL in the config file that specifies what will dynamically loaded, you can easily add new rules that can be used by the Import Process. The following is a simplified definition for the `InvalidIdValidationRule` class. It is for a rule that makes sure invalid ids are not accepted during the import of orders.

```
[ValidationRule("Invalid Id")]
public class InvalidIdValidationRule : ValidationRule{
  public InvalidIdValidationRule() : base() { }
    public override void Validate(ImportContext context)
…}
```

Different rules can be invoked based on client-specified values stored in the database. A common `ImportContext` was passed in that could be used as the context for the new rules. A dynamic tag such as "Invalid Id" could be used for associating the rule in the import language to designate the new rule and when to invoke and run the new rule.

A medical-based AOM system developed by The Refactory for the Illinois Department of Public Health [YJ02] is another example of a system that extensively uses dynamic hook points. In this system, reflection is also used to dynamically bind hook points. Custom behavior can be described as a dynamic method or strategy associated with new types of objects. Thus a new class can be created, and by using reflection, the new behavior can be dynamically associated with new types of diseases and invoked using stored descriptive information.

There are also well-known non-AOM uses of the DYNAMIC HOOK POINTS pattern in the Spring and Eclipse [EIDE] frameworks. They have different implementations with similar intent; the ability to support the definition of hook points and the ability to dynamically invoke new behaviour in well-defined ways.

Acknowledgements

We thank our shepherd Kiran Kumar Reddy for his valuable comments and feedback during the AsianPLoP 2011 Shepherding process. We also thank our 2011 AsianPLoP Writers Workshop Group, Jorge Ortega-Arjona, Gabriele Kahlout, Hironori Washizaki, Masayuki Tokunaga, Norihiro Yoshida, Jonatan Hernández Hernández, for their valuable comments.

References

[AOM] Adaptive Object-Models. http://www.adaptiveobjectmodel.com

[And98] Anderson, F. *A Collection of History Patterns*. Proceedings of the 6th Pattern Language of Programs Conference (PLoP 1998), Monticello, Illinois, USA, 1998.

[Ars00] Arsanjani, A. *Rule Object Pattern Language*. Proceedings of the 8th Pattern Language of Programs Conference (PLoP 2000). Technical Report WUCS-00-29, Dept. of Computer Science, Washington University Department of Computer Science. (2000).

[BMR+96] Buschman, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M. *Pattern Oriented Software Architecture, Volume 1: A System of Patterns*. Wiley & Sons. 1996.

[BR98] Bäumer, D., D. Riehle. *Product Trader*. Pattern Languages of Program Design 3. Edited by Robert Martin, Dirk Riehle, and Frank Buschmann. Addison-Wesley, 1998.

[CW86] Caudill, P., Wirfs-Brock A. "A Third Generation Smalltalk-80 Implementation.", p. 119-130, OOPSLA '86 Conference Proceedings, Portland Oregon, September 29-October 2, 1986.

[EIDE] http://www.eclipse.org/

[EEQX] http://www.eclipse.org/equinox/

[FCW08] Ferreira, H. S., Correia, F. F., and Welicki, L. 2008. *Patterns for data and metadata evolution in adaptive object-models*. Proceedings of the 15th Conference on Pattern Languages of Programs (Nashville, Tennessee, October 18 - 20, 2008). PLoP '08, vol. 477. ACM, New York, NY, 1-9.

[Fow97] Fowler, M. *Analysis Patterns: Reusable Object Models*. Addison-Wesley. 1997.

[Fow02] Fowler, M. *Patterns of Enterprise Application Architecture*. Addison-Wesley. 2002.

[FPR01] Fontura, M., Pree, W., Rump, B. *The UML Profile for Framework Architectures.* Addison-Wesley. 2001.

[FY98] Foote B, J. Yoder. *Metadata and Active Object Models*. Proceedings of Plop98. Technical Report #wucs-98-25, Dept. of Computer Science, Washington University Department of Computer Science, October 1998.

[GHJ+95] Gamma, E., R. Helm, R. Johnson, J. Vlissides. *Design Patterns: Elements of Reusable Object Oriented Software*. Addison-Wesley. 1995.

[HNS10] Atzmon Hen-Tov, Lena Nikolaev, Lior Schachter, Joseph W. Yoder, Rebecca Wirfs-Brock. *Adaptive Object-Model Evolution Patterns*, SugarLoafPLoP 2010.

[HLPS09] Atzmon Hen-Tov, David H. Lorenz, Assaf Pinhasi, Lior Schachter: *ModelTalk: When Everything Is a Domain-Specific Language*, IEEE

Software, vol. 26, no. 4, pp. 39-46, July/Aug. 2009.

[Jon99] Jones, S. *A Framework Recipe.* Building Application Frameworks: Object-Oriented Foundations of Framework Design.Edited by Fayed, M., Johnson, R., Schmidt,.D. John Wiley & Sons. 1999.

[JSFEL] JavaServer Faces Expression Language. http://developers.sun.com/docs/jscreator/help/jsp-jsfel/jsf_expression_language_intro.html

[JW98] Johnson, R., R. Wolf. *Type Object*. Pattern Languages of Program Design 3. Addison-Wesley, 1998.

[JXP] JXPath. http://commons.apache.org/jxpath/

[KJ04] Kircher, M.; P. Jain. *Pattern Oriented Software Architecture, Volume 3: Patterns for Resource Management*. Wiley & Sons. 2004.

[KSS05] Krishna, A., D.C. Schmidt, M. Stal. *Context Object: A Design Pattern for Efficient Middleware Request Processing*. 13th Pattern Language of Programs Conference (PLoP 2005), Monticello, Illinois, USA, 2005.

[Mar02] Martin, R. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall, 2002.

[RFBO01] Riehle, D., Fraleigh S., Bucka-Lassen D., Omorogbe N. *The Architecture of a UML Virtual Machine*. Proceedings of the 2001 Conference on Object-Oriented Program Systems, Languages and Applications (OOPSLA '01), October 2001.

[RTJ05] Riehle D., M. Tilman, and R. Johnson. *Dynamic Object Model.Pattern Languages of Program Design 5.* Edited by Dragos Manolescu, Markus Völter, James Noble. Reading, MA: Addison-Wesley, 2005.

[RY01] Revault, N, J. Yoder. *Adaptive Object-Models and Metamodeling Techniques Workshop Results*. Proceedings of the 15th European Conference on Object Oriented Programming (ECOOP 2001). Budapest, Hungary. 2001.

[SEL] Spring Expression Language. http://www.springsource.org/.

[WYWJ07] Welicki, L.; J. Yoder; R. Wirfs-Brock; R. Johnson. *Towards a Pattern Language for Adaptive Object-Models*. Companion of the ACM SIGPLAN Conference on Object Oriented Programming, Systems, Languages and Applications (OOPSLA 2007), Montreal, Canada, 2007.

[WYW09] Welicki, L.; J. Yoder; R. Wirfs-Brock. *Adaptive Object-Model Builder.* 16th Pattern Language of Programs Conference (PLoP 2009), Chicago, Illinois, USA, 2009.

[WYW07] Welicki, L, J. Yoder, R. Wirfs-Brock. *Rendering Patterns for Adaptive Object Models.*14th Pattern Language of Programs Conference (PLoP 2007), Monticello, Illinois, USA, 2007

[YBJ01] Yoder, J.; F. Balaguer; R. Johnson. *Architecture and Design of Adaptive Object-Models*. Proceedings of the ACM SIGPLAN Conference on Object Oriented Programming, Systems, Languages and Applications (OOPSLA 2001), Tampa, Florida, USA, 2001.

[YJ02] Yoder, J.; R. Johnson. *The Adaptive Object-Model Architectural Style*. IFIP 17th World Computer Congress - TC2 Stream / 3rd IEEE/IFIP Conference on Software Architecture: System Design, Development and Maintenance (WICSA 2002), Montréal, Québec, Canada, 2002.

[YR00] Yoder, J.; R. Razavi. *Metadata and Adaptive Object-Models*. ECOOP Workshops (ECOOP 2000), Cannes, France, 2000.