Bionic Buffalo Tech Note #56

# Asset Assignment Model

## *Abstract*

Various kinds of assets, such as bandwidth, memory, and time, must be allocated and assigned to users as part of the operation of various kinds of applications. The Asset Assignment Model (AAM) describes the entities and actions involved in acquiring, providing, and controlling the use of these assets. Defined and described in UML and IDL, AAM can be used as a general underlying model for specialized applications such as content delivery systems, operating systems, collaborative development environments, and distributed computing environments.

This *Tech Note* briefly describes AAM, and also provides some contrast with complementary and alternative models.

*Publisher's information and notes are at the end of this document.*

## *Contents*

## 1. *Introduction and Rationale*

Various computing environments usually have mechanisms for an application to request the use of assets or resources such as memory, network connections, or CPU time. Sometimes these mechanisms are standardized, sometimes they are proprietary. Examples of standardized mechanisms include:

- The POSIX and UNIX specifications describe APIs to allocate memory (`malloc()`), create file handles (`open()`), or to establish loci of execution (`pthread_create()`).
- CORBA defines factories for various kinds of objects or programming constructs (`Object::duplicate()`, `Container::create_alias()`, or `POA::create_POA()`).
- The DSM-CC User-Network Session protocol defines message sequences enabling a client to request bandwidth, connections, or other resources from the network.

Some protocols or standards make reference to “resources”, but consider different things. For example, the primary focus of the Web Services Resource Framework (WSRF) is communication with “resources”, as they define the term, and not their allocation, assignment, and control.

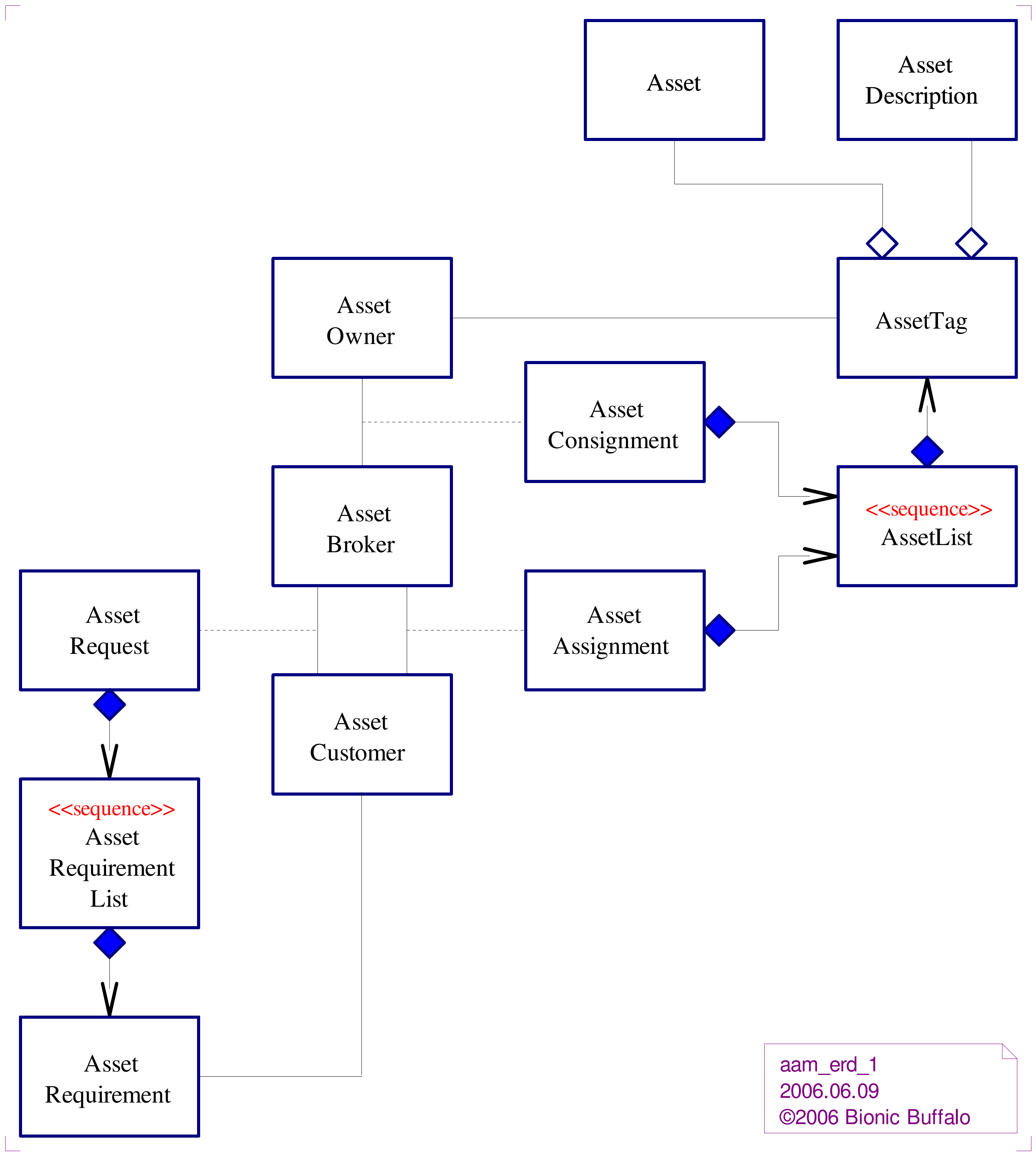
The Asset Assignment Model (AAM) seeks to describe the control, acquisition, and release of generic assets, without defining the behaviour of the assets themselves. The word “asset” is used instead of “resource”, as a better fit to the metaphor and semantics of the model, and to reduce confusion with other models. The model describes the following activities:

- the description of assets in a generalized, extensible way, to facilitate matching assets to requirements
- the aggregation of assets into groups, so that they might be controlled together in atomic operations, without the complexity of a complete transaction oriented model
- the various operations needed to allocate, release, consign, or otherwise control assets in common application environments

AAM is intended not only as an abstract modelling tool, but also is specified in sufficient detail that it might be implemented as a concrete set of objects. It is expected that practical implementations of AAM will create specializations of the abstract AAM objects.

## 2. *Overview of the Model*

The following diagram summarizes AAM.


Asset
Asset Description
Asset Owner
AssetTag
Asset Consignment
<<sequence>> AssetList
Asset Broker
Asset Request
Asset Assignment
Asset Customer
<<sequence>> Asset Requirement List
Asset Requirement
aam_erd_1
2006.06.09
©2006 Bionic Buffalo

An `Asset` is a highly generic entity. Almost any kind of object or entity may be considered to be an `Asset`. Because `Asset`s might be very simple, they are not expected to have any attributes or operations.

`AssetTag`s exist because the `Asset`s themselves are assumed to be only trivial objects. An `AssetTag` has the operations and attributes we might like an `Asset` itself to have, but cannot always expect. In some environments, the `Asset` may not be a “proper” object: it may have a `NIL` reference, but still is presumed to exist, and it can be identified using its description in the `AssetTag`.

The `AssetOwner` is the initial source of `Asset`s. An `AssetOwner` may be a factory, or it may have acquired the `Asset` by transfer from some other `AssetOwner`.

`AssetOwner`s consign `Asset`s to `AssetBroker`s, who attempt to satisfy the requests of `AssetCustomer`s. `AssetBroker`s contribute to the process in two main ways:

- They may consolidate or aggregate the `Asset`s of multiple `AssetOwner`s, so that an `AssetCustomer` may deal with a single `AssetBroker` rather than multiple `AssetOwner`s.
- They may match the requests of `AssetCustomer`s to available inventory, in order to fulfill the `AssetCustomers`' requirements. When a request is satisfied, the `AssetList` is assigned to the `AssetCustomer`.

An `AssetCustomer`, once an `AssetList` have been assigned to him, has use of those `Asset`s until he releases them or until the term of the assignment expires.

## 3. Asset Descriptions and Requirements

An `AssetBroker` must match a customer's requirements to an asset's description. This is done by means of the `AssetDescription` and the `AssetRequirement`.

The `AssetDescription` is a string containing a sequence of assignments of *values* to *variables*. It is essentially the source code for a brief software program. The initial value of the `AssetDescription` is provided by the `AssetOwner`, but it may be modified by the `AssetBroker`.

The `AssetRequirement` is provided by the `AssetCustomer`. It is a short program, in the form of a string, which returns a number (the *score*) indicating the extent to which the description satisfies the requirements. A score of zero is no match at all; a higher valued score is a better match than a lower valued score. The requirement program is evaluated after assigning values to the variables as specified in the description. Undefined variables are given empty or zero values.

The requirement program may also include an assignment to the variable `_alternative_group`. When two or more requirement programs assign the same value to `_alternative_group`, then only the one with the highest score is considered a match. (Of course, none of the available assets may satisfy the requirements.)

## 4. *Consignment*

An `AssetOwner` enables an `AssetBroker` to assign the owner's assets by consigning those assets to the broker. Consignment may be for some limited period of time, or it may be indefinitely until cancelled.

The broker does not request consignment: the process is initiated by the owner. However, a broker may refuse all or part of a consignment. A concrete implementation needing some request by the broker must add new operations to the model.

An entity may inherit both the owner and broker interfaces, being both at the same time.

A consignment optionally may require that the owner be notified in case of assignment.

An owner is not prevented from simultaneous consignment of the same assets to more than one broker. The model includes exceptions for situations where this occurs, or it may be ignored. Simultaneous consignment to multiple brokers may be unwise in many circumstances.

## 5. *Request and Assignment*

The Request-Assignment sequence always is initiated by the `AssetCustomer`.

A special, optional use of an `AssetRequest`, signified by an attribute of the request, causes the broker to return descriptions of all assets meeting the requirements. This acts as a form of advertising on demand. If some kind of active advertising is required in an application inheriting AAM, a third party might make the request of the broker.

Assignments may be indefinite (until cancelled), or for some designated period of time.

If allowed by the broker, an assigned resource may be delegated or reassigned by the customer. The customer may act as a proxy for some other entity. The owner or broker or both may demand to be notified in such cases.

## 6. *Transfer*

An `AssetOwner` may transfer an `Asset` and its `AssetTag` to another `AssetOwner`, which becomes the new owner of the `Asset`. Any brokers having consignments including the `Asset` are notified by the new owner.

## 7. *Pricing and Payment*

AAM does not include explicit provisions for pricing `Asset` use, or for making payments. However, an application can inherit AAM, adding additional features to allow for pricing and payment.

For example, the `AssetRequest` and `AssetDescription` can include pricing variables.

There is no specific provision in the semantics of scoring to allow, say, for a broker to select the cheapest `Asset` meeting some other requirements. Indeed, such problems can become computationally extremely expensive if not entirely intractable. There is nothing in the model, however, to prohibit a specialization of an `AssetBroker` from applying additional policies to selection from among alternatives.

## 8. *Security Policies*

Some security mechanisms are already built into the CORBA used as the foundation for AAM. Such security mechanisms can require that the user of an object must provide some credentials before invoking methods on that object, thus establishing that the user is authorized to invoke those methods. An AAM implementation can employ these mechanisms to restrict access to `Asset`s.

There are some drawbacks to reliance on the CORBA mechanisms for a complete security solution.

- Some `Asset`s might not be implemented in compliant object form, making the CORBA mechanisms unusable.
- Use of the CORBA mechanisms requires support in the infrastructure for features which are not always available. (Minimum CORBA does not support those features.)
- It may be inefficient to defer authentication and authorization until the customer attempts to access the `Asset`.
- Failure to pre-authorize access to an `Asset` might create problems in case of authorization failure.
- The information about an `Asset` found in the `AssetDescription`, including the very knowledge of its existence, might be considered confidential information.
- The outcome of CORBA security decisions is generally binary: “yes” or “no”. There sometimes is a need for a more fine-grained approach, such as granting some customers access to more memory than is available to others.

In order to circumvent these shortcomings, AAM adds some additional mechanisms to be used in conjunction with the standard mechanisms.

- A pre-authorization mechanism is provided. A customer will access a “trial” object to establish whether or not authorization is likely to succeed at the time of use. The trial object also can serve as a surrogate for the `Asset` in case the `Asset` itself does not support common security mechanisms.

- A value type, `AssetUseCriteria`, is defined, with methods to determine usage constraints to be enforced by the broker. A consignment includes a sequence of `AssetUseCriteria`. Each `AssetTag` includes a sequence of zero or more *criteria indices*, each member of which refers to one of the `AssetUseCriteria`. A broker is expected to employ the methods of the `AssetUseCriteria` to enforce the policies they incorporate. Details of this mechanism are beyond the scope of this *Tech Note*; the salient point is that owners may establish usage policies, and expect them to be enforced by the brokers. The scope of the policies may extend beyond security to include other constraints.

Details of these AAM security mechanisms are beyond the scope of this *Tech Note*. For more information, consult the *AAM Technical Manual (Luxembourg)*.

## *Bibliography and References*

*Asset Assignment Model (Luxembourg)*, Technical Manual, Bionic Buffalo

## *Publisher's Information and Notes*

**Publisher.** Bionic Buffalo Corporation, 502 North Division Street, Carson City, Nevada 89703, USA.

**Current Version.** The current version of this document may be found at *http://www.tatanka.com/doc/technote/tn0056.pdf* (for PDF) and *http://www.tatanka.com/doc/technote/tn0056.html* (for HTML).

**Other *Tech Notes*.** For available *Tech Notes*, visit the Bionic Buffalo web site at *http://www.tatanka.com/doc/technote/index.htm*, or e-mail *query@tatanka.com*. Most Bionic Buffalo

*Tech Notes* are available in both HTML and PDF form.

**Key.** PGP/GnuPG key fingerprint: `a836 e7b0 24ad 3259 7c38 b384 8804 5520 2c74 1e5a`.