CAPE-OPEN

Expanding Process Modelling
Capability through Software
Interoperability Standards

M&T Guidelines For Threading

www.colan.org

ARCHIVAL INFORMATION

Title	M&T Guidelines for Threading
Owner	Methods and Tools SIG
Location	https://www.colan.org/wp-content/uploads/2024/05/Guidelines-For-Threading-v1.0.pdf
Document Unique Identifier	9CDE8450-B12C-11EE-9EC1-0800200C9A66
Distribution	Members Only
Status	Draft
Document Version Number	1.0
Created Date	2023-10-24
Revision Date	2024-05-14

Version History

Document Version Number	RFC Date	Release Date	Comments
1.0	2024-02-29	2024-05-14	First published version

IMPORTANT NOTICES

Trademark Usage

Many of the designations used by manufacturers and seller to distinguish their products are claimed as trademarks. Where those designations appear in CO-LaN publications, and the authors are aware of a trademark claim, the designations have been printed in caps or initial caps.

CONTENTS

LIST OF FIGURES

1. Context

Use of multiple threads can improve the performance of computer-aided process engineering applications. Multithreaded software allows for improvements in a wide range of features including user interface performance, faster calculations, simultaneous evaluation of simulation outputs.

The primary driver for initiation of the COBIA project by CO-LaN was to create a middleware platform that was free of limitations imposed by others, in particular Microsoft's Component Object Model (COM). COM has several threading models that constrain the use of multiple threads or reduce the performance of applications by requiring marshaling of calls between objects to synchronize access to different threads when the threading models of the client and server are different.

One of the design requirements for COBIA was compatibility with existing COM components. For this reason, COBIA threading models were designed to be consistent with COM threading models. COBIA provides only two threading models that can be directly mapped to COM threading models, as described below. COBIA provides all necessary support for COM/COBIA interop, including thread synchronization.

Because COBIA is designed to simplify development of CAPE-OPEN objects, and improve performance of multithreaded applications, COBIA is recommended for use in applications and components relying on CAPE-OPEN for interoperability.

1.1 Terminology

Apartment: Microsoft term for a "logical container" that creates an association between objects and, in some cases, threads. Threads are not apartments, although there may be a single thread logically associated with an apartment in the Single threaded apartment (STA) model. Objects are not apartments, although every object is associated with one and only one apartment.

Application: a computer program designed to carry out a specific task other than one relating to the operation of the computer itself [Wikipedia]. A process modeling environmental (PME) is an example of an application.

Client: a software object that uses services. Typically, a PME is a client utilizing services provided by a PMC. Client is defined here to help align terminology used by Microsoft in COM documentation with CAPE-OPEN usage.

Component: a software object used by an application.

In-process: a PME and associated PMC instances running in one or more threads on the same computer utilizing shared resources. An in-process application can access external components that run on multiple threads using system resources controlled by the program.

Multithreaded: use of more than one thread by an instance of an application.

Object: the general term for an application or software component, such as a PME or PMC, developed using object-oriented programming principles.

Out-of-Process: components used by the application that run in a separate, independent program. Out-of-process components run in different tasks and use separate resources from the main application. Out-of-process components are responsible for synchronizing their state.

Process: is a collection of virtual memory space, code, data, and system resources. An instance of an application runs in a process.

Process modeling component (PMC): a CAPE-OPEN component used by a PME provided as a service.

Process modeling environment (PME): a CAPE-OPEN application that uses services provided by a PMC.

Remote: access to resources that are located on a different computer than the one hosting the running process.

Service: a component accessed by a client. Typically, PMCs provide services that may be accessed by a PME. Service is defined to help align terminology used by Microsoft in COM documentation with CAPE-OPEN usage.

Thread: code that is to be serially executed within a process. A processor executes threads, not processes, so each application has at least one process, and a process always has at least one thread of execution, known as the primary thread. A process can have multiple threads in addition to the primary (main) thread.

Thread safe: applications and their components accessed by them manipulate shared resources in a manner that ensures that the application and components behave properly and fulfill their design specifications when accessed concurrently by multiple threads.

2. Purpose

The purpose of this document is:

1. Design COBIA's requirements for threading and the implications of these requirements on CAPE-OPEN,
2. Present an overview of the COM in-process threading models as they relate to CAPE-OPEN applications and component development,
3. Present the design of the COBIA threading models,
4. Describe the interactions between objects using the COBIA and COM threading models,
5. Provide best practices for the use of COM's threading models on component development to improve performance and facilitate the transition of COM components to COBIA,
6. Disseminate this design and advise CAPE-OPEN software vendors on which actions to take to modernize their implementations.

3. Use of Multiple Threads in CAPE-OPEN Applications.

Development of multithreaded applications can be problematic, and if not done properly, can result in decreased performance, incorrect results, or application crashes. When designing multithreaded applications, one must take care not to inadvertently change the state of the PMC concurrently from multiple threads. This section considers how multiple threads in the PME can access these shared resources in the PMC without causing data races.

3.1 Establishing and Maintaining the State of a PMC

The state of the PMC is anything owned by the PMC that can be accessed or modified. Internally, PMCs contain resources, internal information, and secondary objects owned by the PMC. Externally, the PMC has relationships to objects that are provided by the PME, such as the simulation context, material objects. In sum, the internal and external elements of the PMC establish the state of the PMC.

In a multithreaded scenario, care must be taken that the state of PMC is not modified during execution of one or more other calls that depend on the state. In the case of concurrent calls, this would lead to a race condition for which accessing the state from one thread while modifying it from another thread leads to unpredictable results.

For non-concurrent access to the PMC, calls from one thread might inadvertently interrupt the sequence of calls from another thread. For example, Thread A executes a sequence of calls to set the pressure, temperature and composition on the material object associated with a property package and then requests the property package to perform a single-phase property calculation. If during this sequence of calls, Thread B modifies, for example, the temperature of the material object associated with the property package (or associates a different material object with the property package), the value of the single-phase property obtained by Thread A will be inconsistent with the conditions imposed by Thread A. The PME is responsible for ensuring that the states of objects associated with one another remain consistent throughout the sequence of calls during which this association exists.

The PME must also ensure that the external elements of the state of the PMC are managed when accessing the PMC from multiple threads. For example, the sequence of connecting ports and calculating a Unit Operation from one thread should not be interrupted by modifying the port connections of the Unit Operation PMC from another thread. These conflicting operations could create a race condition, possibly causing the application to crash.

A PMC should in principle let the PME be in charge of division of calculation load between threads. A PMC is usually considered a smaller part of a bigger calculation system and is not in the best place to optimize load balancing over the threads. From this perspective it is advised that PMCs should consider that multithreading is not as advantageous as it would be in case the same calculation is performed by the same software in a stand-alone environment. PMCs should run on a single thread, unless the PMC has good reasons to assume that it is one of the most calculation intensive PMCs in the flowsheet (for example CFD based unit operations). In order not to affect the level of thread safety of the PME, a multithreaded PMC must restrict all PME callbacks to the thread on which the PME called the PMC.

3.2 Requirements

The following general requirements are formulated for use in both the COM and COBIA.

Requirement 1 (PME): The PME must ensure that the external elements of the state of the PMC are managed when accessing the PMC from multiple threads.

Rationale: This requirement protects the external state of the PMC from concurrent access from multiple threads. For example, the PME should not change the value of a parameter or port connection during the calculation of the PMC.

PMCs should not be forced to be more thread safe than the thread safety imposed on them by the COM or COBIA threading model they choose to support.

Requirement 2 (PMC): A PMC may not access PME objects from threads that the PMC has itself started.

Rationale: unrestricted access of PME objects by a PMC would require the PME to be prepared for incoming calls from foreign threads on its own objects, and this is not a requirement that CAPE-OPEN should impose on PMEs.

Note: Should the use case arise, this requirement may be loosened by defining in a future CAPE-OPEN version an interface, implemented on the material object, allowing the duplication of a material object on different threads.

4. COM Threading Models

A COM apartment provides a means of thinking about concurrency and how threads relate to a set of COM objects. A COM apartment is a subdivision of a process used to create an association between objects and, in some cases, threads. Neither objects nor threads are apartments, but threads can be assigned to an apartment, and an object can live in one, and only one apartment. Using the concept of an apartment, COM provides mechanisms that allow applications (process modeling environments or PMEs) to use components (process modeling components or PMCs) that use different threading models to work together and provides synchronization services when required.

This discussion provides an overview of COM threading models, particular as they relate to use within the CAPE-OPEN development paradigm. References below are made to Microsoft and other documentation that can be review for further information.

4.1 Use of Multiple Threads in COM by a Process Modeling Environment (PME)

A PME initializes COM on each thread where the PME will use COM. COM provides two options for threading by an application, such as a PME, a single threaded apartment (STA), or a multithreaded apartment (MTA).

One approach to multithreading by a PME is through the use of multiple STAs. A PME can initialize and use multiple STAs. This would allow the PME to create PMCs on a particular thread and the PME can only access that PMC through that thread. The PME can make calls to the PMC from any of these STAs but will need to synchronize calls to the PMC by the PME. In addition, for PMCs that are not "Apartment" or "Both" thread model, COM will marshal all calls to the PMC.

Alternatively, the PME can initialize and use one MTA for multithreading. When the first MTA thread is initialized, the MTA is created for the application. Once the MTA has been created, when additional MTA threads are created, they are assigned the same MTA. A process can simultaneously have any number of STAs and at most one MTA. The PME can make calls from the MTA to any PMCs that were created from an MTA thread. Note that for all PMCs that are not "Free" or "Both" thread model, COM will marshal all calls from the MTA to the PMC.

The threading design chosen for the PME affects the requirement to marshal calls to PMCs instantiated by the PME. Threading models for PMCs are described below, along with marshaling requirements based upon the type of Apartment the PME is using and the threading model of the PMC.

4.2 COM Component Thread models

In-process components (PMCs) used by an PME can be called from one or more STAs, or one or more threads that belong to the MTA. COM in-process server (InprocServer32) threading models are defined by the `ThreadingModel` registry entry. The following is a list of `ThreadingModel` registry entries that can be use by an in-process PMC:

1. No `ThreadingModel` registered: A component registered without a threading model will be created in the main single-threaded apartment of an application and the component and its entry points may only be accessed through the main thread.

2. "`Apartment`" `ThreadingModel`: "`Apartment`" threaded components that may only be accessed from the STA that created it, but multiple instances of an "`Apartment`" threaded components can be

created in the same process by different STAs. "Apartment" ThreadingModel components must ensure that their entry points and any global data are thread safe. Calls to "Apartment" ThreadingModel object that originates from a different STA than the one that created the object, or from an MTA are marshaled.

3. "Free" ThreadingModel: An object that implements its own synchronization and is thread safe for access from multiple threads at the same time. A "free" threaded object may not be compatible with the STA model, so calls from an STA are marshaled to the object. "Free" threaded components may be simultaneously called from any thread, can pass interface pointers directly to threads it created, and can be called on these pointers without marshaling.

4. "Both" ThreadingModel: A "Both" threaded object implements its own synchronization and can be access concurrently by multiple apartments. Depending on the thread in which the object is created, it will behave as "Apartment" or "Free" threaded.

5. "Neutral" ThreadingModel: A COM+ thread model that is typically not suitable for CAPE-OPEN applications. Neutral threaded PMC executes on the neutral thread, of which there is only one. All access to the neutral thread is marshaled.

The choice of initializing COM using single- or multithreaded apartment(s) for the application and the ThreadingModel of the PMC effects actions taken by Windows to synchronize threads between the PME and PMC. When the PME creates an in-process COM PMC, windows checks whether the PMC's threading model is compatible with the apartment to which the creating thread belongs. If so, the PMC is created in the current thread and can be called directly from the current thread (no marshaling is applied). Otherwise, Windows will create the PMC in a COM provided compatible thread and marshaling applies. An object that is capable of both MTA and STA threading (ThreadingModel = 'Both') is always compatible with the current thread but will behave for its lifetime as an STA or MTA object, depending on the apartment that the current belongs.

Most COM-based CAPE-OPEN implementations use the COM apartment threading model because it enforced resolution of the issues through COM's synchronization of these states. However, this results in decreased performance because COM imposes its synchronization rules on the interactions between property package, material context and unit operation states.

For "Out of Process" COM components, marshaling is always applied and it is always synchronized over a single thread.

4.3 Recommendations for Use of COM Threading Models in CAPE-OPEN

The following sections provide recommendations for use of multiple threads by PMEs and the selection of COM threading model for PMCs.

4.3.1 PME Thread usage considerations

Use of incompatible thread models for instantiation of a COM component forces COM to protect the thread safety of the component by marshaling the component. Unnecessary use of marshaling results in reduced performance.

As described above, multithreaded COM PMEs have the option of creating multiple STAs, or a single MTA that manages multiple threads. The choice of which model to choose can affect the performance of the application because COM will marshal calls from the PME's MTA thread to a STA PMC, and calls from a PME STA thread to a 'Free' threaded PMC. The PME can make calls from either the MTA or an STA without marshaling PMCs that are registered using the "Both" threading model. For this reason, it is advisable that PMCs are registered under the "Both" threading model. Note that this requires that the PMC to have a level of thread safety that corresponds to "Free" threading. This can be accomplished, for example, by a per-PMC instance of a C++ mutex that guards all COM-based access to the PMC.

4.3.2 Threading Model Selection for PMCs

PMC developers need to properly register their components based upon their ability to protect the internal state of their component. For PMCs that make certain that each instance of a PMC is thread safe.

Use of the 'STA' ThreadingModel restricts access of the PMC to the thread that it was created on, but because multiple instances of the PMC can be created on different threads, the PMC must protect the state of each instance of the PMC from other potential instances. To be thread safe, STA threading model PMCs should avoid the use of global variables or COMMON blocks that can be accessed from different instances of the PMC running on different threads. If COMMON blocks or global variables are used, they must be protected against concurrent access from multiple threads.

The 'Free' threaded model allows the PMC to be called from any thread in the PME's MTA, so it should be prepared to deal with concurrent incoming calls.

The most flexible option for development is to allow the PMC to meet both the requirements for use by both an STA and an MTA (ThreadingModel = "Both"). This approach is recommended for the development of COM-based CAPE-OPEN PMCs as

- it allows unmarshalled access to the PMC when the PME initializes COM either in an STA or MTA,
- the PMC provides its own synchronization in case the PME initializes COM in an MTA.

4.4 Further information about COM

For further information about threading under COM, please refer to the following websites:

https://stackoverflow.com/questions/127188/could-you-explain-sta-and-mta

https://docs.microsoft.com/en-us/windows/win32/com/processes--threads--and-apartments

https://docs.microsoft.com/en-US/troubleshoot/windows/win32/descriptions-workings-ole-threading-models

https://devblogs.microsoft.com/oldnewthing/20151020-00/?p=91321

https://docs.microsoft.com/en-us/windows/win32/cossdk/neutral-apartments

5. COBIA Threading Models

The considerations for the development of the COBIA threading models are twofold: first, the threading models are designed based upon modern threading approaches, so they are flexible for future approaches to multithreaded development; and secondly, to provide a framework for interoperability with the COM threading models to ensure compatibility with legacy CAPE-OPEN components and applications.

The PME chooses a Threading Model for each PMC upon creation of the PMC. The Threading Model may differ from one PMC to the other. This contrasts with COM where the threading model is chosen per thread or apartment, and not per PMC.

This chapter gives the requirement imposed on the design of COBIA Threading Models, followed by the description of the COBIA Threading Models themselves.

5.1 Requirement

The following general requirement is formulated for use by COBIA-based PMEs.

Requirement 3 (PME): The PME must not concurrently access a primary PMC object or any of its secondary objects, even for PMCs that advertise as multithreading capable.

Rationale: Although in COBIA we want to allow the use of PMCs from multiple threads, allowing for concurrent calls from multiple threads substantially complicates PMC development. Rather than place the burden of internal synchronization on each PMC, the requirement for synchronizing lies with the PME.

The implication of some concurrent actions prevented by the above requirement is obvious: connecting or disconnecting a unit operation port, or changing a parameter value, while a unit operation is calculating, is clearly undesired. Not as obvious is that this requirement also implies that the PME cannot monitor a parameter value or obtain the unit's name concurrently with any other method call. This requirement is needed to prevent the unit from internally shielding its state from race conditions.

5.2 Definition of COBIA Threading Models

COBIA defines two threading models for interactions between the PME and PMCs.

- `Default`: A PME can invoke methods in PMCs from any thread, subject to Requirement 3. The COBIA '`Default`' model is designed to allow a PMC to be accessed from multiple threads by a PME without the need for the PMC to be concerned with it being concurrently accessed from multiple threads. As opposed to the COM '`Free`' `ThreadingModel`, synchronization is the responsibility of the PME. PMCs registered using the '`Default`' model are never marshaled in COBIA.

- Restricted: The PME may invoke a PMC only on a single thread, the one in which the PMC was created. COBIA 'Restricted' model is like the COM '`Apartment`' `ThreadingModel`. PMCs advertising the 'Restricted' model will be marshaled when created by the PME with the '`Default`' model. PMCs created for access compatible with the 'Restricted' model are never marshaled by COBIA. At PMC creation, the PME may indicate that the access to the PMC is compatible with the 'Restricted' model.

A PMC advertises its threading capabilities through its registration in the COBIA registry. At creation time of a PMC, a PME indicates whether access by the PME to the PMC complies with the restrictions implied by the 'Restricted' model or not.

The COBIA threading models are intended solely for COBIA-mediated interaction between software components, and do not prohibit the use of multiple threads by either the PME or the PMC. 'Restricted' threading is not the same as single-threaded because the PME may be multithreaded, and multiple instances of the same PMC type may be created on different PME threads. As a result, even PMCs advertising the 'Restricted' model may need to provide protection against thread safety violations. For instance, any use of global or common block variables must be protected from race conditions through synchronization.

COBIA's only constraints on multiple threads are those stated for the selected threading model.

5.2.1 Registration and Instantiation Flags

COBIA utilizes registration and instantiation flags to indicate which set of rules the PME will need to comply with while interacting with the PMC. A PMC should be registered as requiring 'Restricted' threading if it is only able to comply with 'Restricted' threading model.

When registering a PMC, the `CapePMCRegistrationFlags` enumerated type is used to advertise the threading support of the PMC.

CapePMCRegistrationFlags		
Named Value	**Integer Value**	**Description**
CapePMCRegistrationFlag_None	0	No specific options. The PMC must be capable of '`Default`' threading
CapePMCRegistrationFlag_RestrictedThreading	1	PMC requires 'Restricted' threading

By default, an object registered using the `CapePMCRegistrationFlag_None` is instantiated unmarshalled, in the current thread. This allows the PME to access the PMC non-concurrently from multiple threads.

A COBIA PME invokes the `COBIACreateInstance` method to instantiate a new instance of a COBIA PMC. The `COBIACreateInstance` method has a flag parameter to specify the intent of PME's access to the PMC instance.

CapePMCCreationFlags		
Named Value	**Integer Value**	**Description**
CapePMCCreationFlag_Default	0	No specific options. The PME may access the PMC non-concurrently from multiple threads.

CapePMCCreationFlag_AllowRestrictedThreading	1	The PME promises to access the PMC only from the current thread.

PMCs registered with the `CapePMCRegistrationFlag_RestrictedThreading` value will be marshalled by COBIA if the PME does not specify the `CapePMCCreationFlag_AllowRestrictedThreading`. To prevent unnecessary marshalling, PMEs that access PMCs only from the current thread, should specify `CapePMCCreationFlag_AllowRestrictedThreading`.

PMC objects created by other PMC objects inherit the threading model of their parent.

5.2.2 Use of a Manager Object

PMCs created through a Manager, for example Property Packages created through a Property Package Manager, do not have their own registration. In this case, each PMC has the threading model advertised through the Manager.

Two PMCs created from a Manager advertising 'Default' threading may be accessed concurrently, since after their creation they are both independent, Primary PMC objects. Note that for each PMC, requirement 3 still holds.

PMCs created from a Manager advertising 'Restricted' threading are created in the same thread as the Manager and can only be used from that thread. To create two PMCs of the same type from a Manager in different threads, unmarshalled, the Manager itself must first be created in each thread specifying `CapePMCCreationFlag_AllowRestrictedThreading`.

5.3 Marshaling of calls

Calls between objects are marshaled in COBIA when a 'Restricted' threading model PMC is created by the PME requesting the 'Default' threading model.

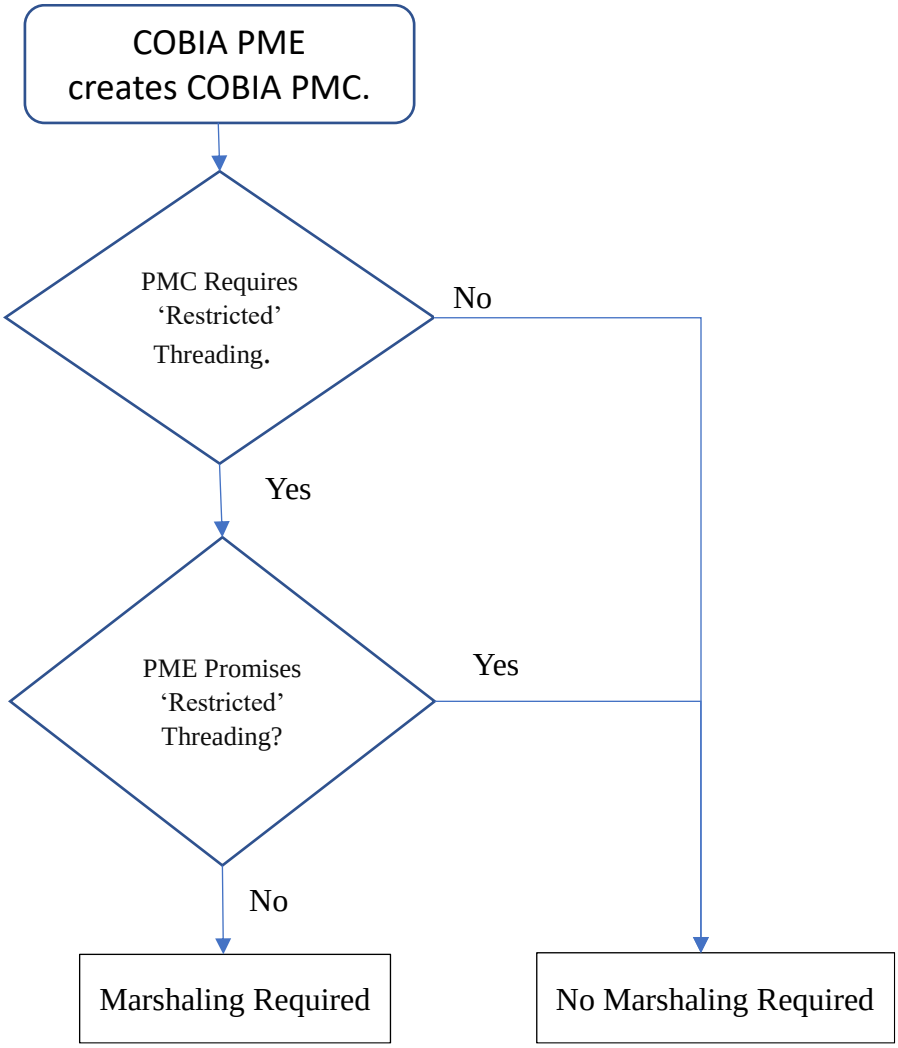

Figure 1. COBIA-specific Marshaling decision tree based upon choice of COBIA thread models.

6. Threading and COM-COBIA interoperability

This section describes registration of COBIA PMCs for use by a COM PME, use of COBIA PMCs by a COM PME, and use of COM PMCs by a COBIA PME. COBIA handles all the interactions with COM, including identifying the need to marshal calls between the PME and PMC.

6.1 Access COBIA PMCs from COM PME

When registering a PMC in the COBIA registry, COBIA also registers the PMC for COM interoperability as follows:

- PMCs that are registered without the `CapePMCRegistrationFlag_ RestrictedThreading` flag, are registered with the '`Both`' COM threading model.
- PMCs that are registered with the `CapePMCRegistrationFlag_ RestrictedThreading` threading model `CapePMCRegistrationFlags` are registered with the '`Apartment`' COM threading model.

In this manner, all COBIA threading restrictions are satisfied, except that the COM PME may concurrently access a PMC that is created in COM MTA mode. This is in contradiction with Requirement 3 (PME). COBIA works around this by synchronizing access to the COBIA PMC and all its secondary objects.

All required marshaling, if any, is done automatically by the COM machinery.

6.2 Access COM PMCs from a COBIA PME

Prior to creation of a COM PMC, COM must be initialized on the thread on which the PMC will be created. COBIA must not initialize COM on any COBIA PME thread for the following two reasons:

- Once COM is initialized and a COM threading model is selected, the COM threading model cannot be changed on that thread. As COBIA is unable to tell what the COBIA PME requirements are for COM initialization, COBIA is unable to decide what is a reasonable COM threading model for the COBIA PME thread.
- If COM is initialized on a thread, it must also be un-initialized on the same thread. There is no reasonable time at which COBIA can un-initialize COM on a thread: COM cannot be un-initialized during the DLL_THREAD_DETACH notification in the DllMain function according to Microsoft requirements. Moreover, the DLL_THREAD_DETACH notification may not take place for all threads. There is no COBIA-defined function that must be called by the PME prior to terminating a thread.

Therefore, COBIA only initializes COM on threads that are created by COBIA itself. In this scenario, all COM PMCs are created on COBIA threads, and COBIA access to COM PMCs is marshaled over COBIA threads.

To avoid this, the PME may initialize COM appropriately on its own thread(s). COM un-initialization is then also a PME responsibility. Now COBIA can create COM PMCs directly in the COBIA thread.

COBIA PMEs that predominately access PMCs according to the rules of the 'Restricted' threading model may be better served initializing COM using the STA threading model. In this case, 'Apartment' and 'Both' threaded COM PMCs can be used on the COBIA PME's thread without marshaling, provided that the COBIA PME requests creation of the PMCs for 'Restricted' threading

(`CapePMCCreationFlag_AllowRestrictedThreading`). Single-threaded COBIA PMEs should follow this pattern.

COBIA PMEs that predominantly access PMCs according to the rules of the 'Default' threading model may be better served initializing COM using the MTA threading model on all the COBIA PME's threads that create and use PMCs. In this case, 'Free' and 'Both' threaded COM PMCs can be used on the PME's thread without marshaling. In this scenario, 'Apartment' threaded PMCs will be marshaled.

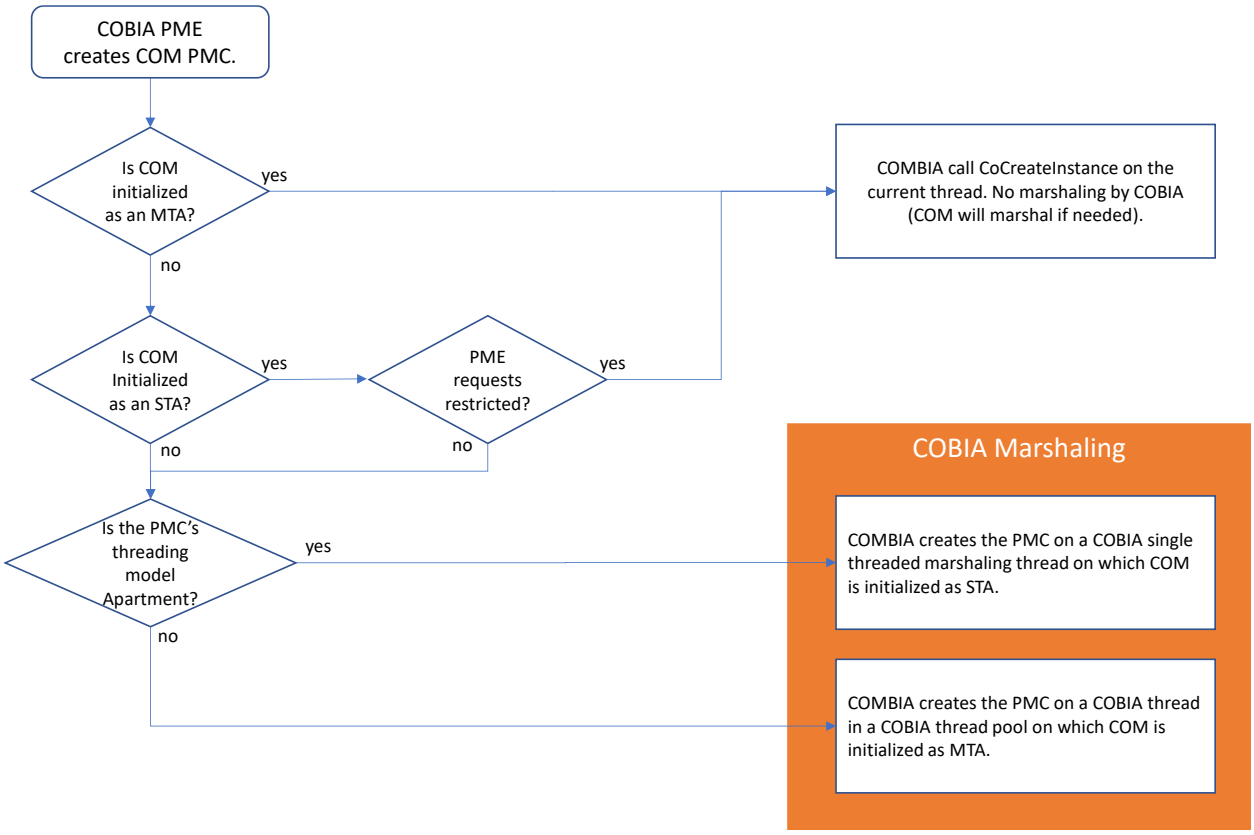

Figure 2. Decision process applied by COBIA for the Instantiation of a COM-based PMC.