Analysis of Approaches for Supporting the Open Provenance Model: A Case Study of the Trident Workflow Workbench
Yogesh Simmhan and Roger Barga
{yoges, barga}@microsoft.com
Microsoft Research, Redmond WA 98052

Abstract
The Trident workbench is a platform for composing, executing and managing scientific workflows. While Trident collects provenance in its native provenance model, the third provenance challenge was an opportunity to build support for the Open Provenance Model into Trident. There are several possible approaches to harmonize our native model with OPM, and such choices are also available to other existing provenance and workflow systems working towards OPM compatibility. We identify and analyze the relative merits of these approaches in an effort to inform practitioners planning to support OPM in their existing provenance/workflow systems. Further, we describe our experience with using the integration approach we choose to interoperate with other teams as part of the challenge.

1 Introduction
The Trident scientific workflow workbench [1] is an open source workflow platform from Microsoft Research for composing, orchestrating, and managing in silico science applications as data and control flows. EScience workflows are typically data intensive and provenance – the derivation history of data products – is important for verification, reproduction, and informed reuse of data used and produced by such workflows [2, 17]. While provenance collection for scientific workflows is well studied [3, 4, 5, 6], efforts to share provenance recorded from different workflow systems have been hampered by the lack of agreement of what constitutes provenance and a common model for it. The provenance challenge series [7] has facilitated efforts to reach consensus on a specification for provenance, where participants record and share provenance for a defined problem, usually a workflow, and attempt to answer queries over the provenance collected and imported. The series have active engagement by the scientific workflow and provenance community.

Third provenance challenge [8] saw the first, broad use of the Open Provenance Model (OPM) specification [9]. We built compatibility with the OPM specification in the Trident workflow workbench as part of attempting the challenge. Trident collects and records provenance in its native provenance model from scientific workflows and it is stored in a relational, local XML or cloud storage accessed through a data abstraction layer. Different approaches were considered for integrating OPM support into Trident, ranging from adopting OPM as our native provenance model to keeping them decoupled with thin import-export libraries for exchanging OPM graphs as needed. Such choices are equally applicable to other existing workflow and provenance systems considering supporting OPM. Both practical and technical considerations drive the choice, and the selection has an impact on the performance and interoperability features of the resulting system.

This article discusses our efforts to harmonize the native provenance collection in Trident with OPM and our analysis of the approaches considered before opting for one to attempt the provenance challenge. Our experience helps guide existing workflow and provenance systems that use non-OPM provenance natively to migrate to the specification and to achieve interoperability with other provenance systems.

Prior challenges provide discussions of provenance features present in different provenance systems [7]. Literature in this special issue and beyond examines how to support OPM in individual provenance frameworks [15, 18]. Many provenance systems achieve interoperability through schema mapping and information integration without exploring possible modifications to their own system [22]. For e.g., Ellkvist, et al., have proposed a mediation architecture to provenance interoperability that uses a global schema, similar to OPM, specialized for scientific workflows that is similar to one of the approaches we investigate [21]. While useful research contributions, they provide limited practical guidance for architects of provenance systems who are considering changes to become compatible with a common provenance specification like OPM [24]. There is no work to our knowledge of a detailed analysis of implementation approaches and challenges to integrate OPM into an existing provenance and workflow system.

This article addresses this lacuna and makes the following specific contributions:
1. We introduce and analyze four different approaches to support OPM in Trident, generalizable to other existing workflow and provenance management systems.
2. We describe our provenance challenge attempt to achieve interoperability using the approach we select for OPM integration in Trident.
The rest of this article is organized as follows: we introduce the Trident workbench in Section 2, describe Trident’s native provenance framework in Section 3, propose and analyze approaches to support OPM in Trident in Section 4, demonstrate its application to the provenance challenge for interoperability in Section 5, and present our conclusion in Section 6.

2 Trident Scientific Workflow Workbench

The Trident workbench [1] is a platform for composing and executing scientific workflows, developed at Microsoft Research as an open source tool for running scientific applications and for workflow research. Trident uses the business-class .NET Windows Workflow engine [10], which accompanies Windows installations, for orchestration while adding the necessary middleware services and composition interface to support scientific workflow management.

The Trident composer allows building of workflows with control and data flows from built-in and user-defined activities, as well as nested sub-workflows. Activities provided include control flow operations, common data formatting for eScience, web service invocation, and SQL querying; additional user-defined .NET activities for specific science domains can be written or imported. Workflows are represented in XOML (an XML format) while activities are packaged in DLL libraries.

Data flow designates data dependencies between activities such that an activity can execute only when all its inputs, potentially from other activities, are available. Control flow activities, such as sequences, conditions and (parallel) iterators, enforce temporal ordering between activities such that an activity cannot execute until a previous activity completes. One implication of this for provenance collection is that while an activity may not have receive input data from a previous (control flow) activity output, the execution of that activity depends on the completion of the prior control flow activity.

The Trident workbench can launch workflows on the local machine, a remote machine, or an HPC cluster. An Execution Service running on the target machine or HPC head node queues execution requests, stages workflow dependency binaries in the local working directory, and monitors workflow runtime progress through distributed events published by the workflow engine and activities using the Blackboard publish-subscribe infrastructure [12].

The Trident Registry is a shared repository for users to store, manage and version workflows, activities, and libraries. It also records metadata about resources available for workflow execution, such as HPC clusters, user and group permissions, and handlers for accessing external data sources. The workflow runtime monitoring and provenance metadata are also recorded and managed in the Registry. The Registry API provides object-to-storage bindings based on a flexible data model. It allows users to configure a schema for the data organization that includes entities, typed attributes, and relationships, with .NET objects generated for them. The schema is mapped to different storage types, such as a relational database, XML file, or cloud storage, using storage adapters. Queries are performed using LINQ, a storage independent, type-safe, .NET query language abstraction.

Trident is similar to other existing scientific workflow systems in many respects, and the issues of provenance collection and OPM integration that we encounter are likewise relevant for other existing workflow systems.

3 Native Provenance Collection in Trident

Trident supports provenance collection as part of its runtime monitoring infrastructure that builds on top of Windows Workflow engine features [20]. The engine automatically generates events that trace the start and stop of a workflow, the activity execution, and exceptions generated during workflow orchestration. In addition to the execution events from the engine, Trident tracks data flow by generating events that record the input and output data to the activities. These events can publish just the reference to the data or the entire data object itself, including collections.

The above in-memory events are published from the local machine where the workflow executes to a persistent Registry on a different server using the Blackboard pub-sub system. Different notification clients subscribe to workflow events, including, for recording provenance in the Registry and displaying workflow progress in the Composer. The events contain name-value pairs that record metadata like the workflow instance and activity that generated the event, status, timestamp, and names and values of input/output data for the activity generating the event. The Execution Service subscribes to status notifications generated by workflow instances, and ensures that the workflow completes successfully — potentially relaunching the workflow or taking other policy action if the execution fails.

The Provenance Service records and analyses the workflow events to synthesize a provenance graph out of it, stored in a native provenance model within the Registry. User can visualize provenance and other workflow metadata from the workbench, allowing them to inspect every aspect of the workflow execution at any time, looking at the inputs and outputs to each activity, their execution times and statuses, their resource usage and even the DLLs used for the execution.
Different granularities of workflow tracking allows both simple monitoring through status events and holistic provenance recording by the provenance service.

Figure 1 shows the data model used in Trident to record provenance (green) and workflow composition (red) metadata in the Registry. The Trident native provenance model is linked with the available workflow composition. The composition metadata provides static, structural information about the activities and control/data flows that are present in a workflow during workflow composition. For example, the BoundTo attribute of the Param Assignment entity denotes an input to an activity, and it can be assigned to the output parameter of a different activity – a data flow – as indicated by the dependency arrow. Similarly, the Activity Sequence entity is used for representing control flow. Additionally, runtime status information about the workflow and activity instances are represented using the four provenance entities that form Trident’s native provenance model, and are populated by the execution service. These record the input and output data values and types for each activity (InputOutput Param entity), their order and time of execution (Processing Status entity), and information about the activity and workflow instances being executed (Provenance Info entity).

The provenance entities have limited dependency on other metadata in the Registry to make provenance handling modular and extensible in future. However, the existing relationship between runtime provenance information and the workflow structure permits complex queries that depend on control flow and better interpret the runtime information. The single dependency arrow between Provenance Info’s Job ID attribute and Job’s ID attribute allows for these queries to be natively answered using the Registry’s LINQ query mechanism.

4 Supporting the Open Provenance Model in Trident

The provenance information collected in Trident is comparable to the Open Provenance Model. In fact, the native provenance model along with the composition metadata in Trident is a superset of the OPM specification. While this makes it easier to map the Trident provenance schema to OPM for interoperability with external provenance systems, there are different approaches with associated advantages and disadvantages to perform this integration.

4.1 Approaches to OPM Integration

It is possible to classify approaches to support OPM based on the extent to which OPM is integrated into Trident, both at the schema and the implementation levels. Given the resemblance of Trident to existing scientific workflow systems, these approaches are generalizable to other workflow and provenance systems. These approaches are:

A. **Tightly Coupled Integration**: In this case, we fully change the native provenance model in Trident to OPM (Fig. 2(a)). All aspects of provenance, including collection, storage, query, and presentation, use OPM terms and semantics. This ensures a consistent and uniform provenance concept in the Trident workbench.

B. **Loosely Coupled Storage Integration**: Here, we store provenance using both Trident’s existing model and in the OPM schema. Since the Provenance Service populates the native Trident provenance model from runtime events, the service can also do the same to map the events to OPM and record it in the OPM store (using the Registry) in realtime (Fig. 2(b)). This allows OPM graphs to be imported into the OPM store but not the Trident native provenance store.

C. **Loosely Coupled Query Integration**: When the Trident Registry uses a SQL backend for storing provenance, it is possible to build database views that match the OPM data model on top of the native provenance model. This allows a single provenance copy to provide multiple views that support queries using both Trident and OPM schemas (Fig. 2(c)). However, the use of OPM views (rather than tables) restricts the ability to import external OPM graphs into the store.

D. **Fully Decoupled**: In this approach, we store OPM data in a completely separate store that uses the OPM schema. The native Trident schema continues to be recorded as before. Import and export tools transfer provenance data between OPM and Trident provenance stores, while queries can run on the two data models independently (Fig. 2(d)).
4.2 Analysis of Approaches

We analyze the different approaches based on performance and overhead measures, qualitative factors and interoperability features. While this analysis uses the Trident workflow and provenance collection system as an example, the discussion is equally applicable to other practitioners of provenance who are considering support for OPM.

4.2.1 Coding Effort and Maintenance

The effort required to implement each of the four approaches and maintain the code vary significantly. This is crucial for a project like Trident where project delivery deadlines need to be met and the shared source code and architecture should be understood and updateable by the developer community at large.

The most code intensive approach is tight integration since this permeates through all aspects of the Trident system. Attributes in provenance collection events need to use OPM terms, potentially affecting user-defined activities that may generate some of these events. The native provenance model in the Registry has to be modified, causing changes to existing Trident operation queries that depend on the model. The GUI displaying provenance graphs and metadata has to reflect the new model. Also, the OPM specification was not final before the challenge and keeping the code compatible with a changing specification is burdensome. The advantage of this approach, however, is that there is just one well-specified provenance model used consistently within Trident for developers and code maintainers to be aware of.

The storage integration approach has lesser code overhead as it localizes the coding impact to the provenance service that populates the provenance model from runtime events. While mapping runtime events to the OPM schema is non-trivial, it does not require extensive refactoring of the codebase. The OPM provenance store is separate from the Trident provenance store, so existing Trident GUI components and provenance queries are not affected. Changes to the OPM specification also involves updating just the OPM mapping module.

The query integration approach has the least coding burden since it requires just the definition of SQL query views based on the native provenance model, which is arguably simpler than the other approaches. Also keeping consistent with the OPM specification requires just maintaining the view query. However, the view query needs to be updated even if Trident’s native provenance schema changes.

The decoupled approach has limited impact on the existing codebase but requires additional code to import and export provenance documents into and out of the native Trident and the OPM stores. Changes to either schemas also require updation of the import-export modules.

4.2.2 Timeliness and Data Consistency

The approaches provide varying degrees of consistency between provenance available in the Trident model and OPM, and in the timeliness of provenance availability. These affect the use of provenance for, say, auditing systems that track data usage in realtime or information integration systems that analyze provenance from across multiple provenance stores.

In the tight integration approach, there is absolute consistency since there is only one copy of the provenance that arrives from the Trident workflows or is imported from external provenance systems. Provenance is also available in near realtime since the provenance events are stored in the OPM data store as soon as they form a complete provenance graph.
The storage integration approach too provides timeliness since the provenance events are inserted into both Trident and OPM data stores as soon as available. However, there may be boundary cases where the provenance metadata available in the Trident store may be fresher than the OPM store. This is because provenance events are buffered in memory until they form a complete and consistent OPM graph, after which they are inserted into the OPM store; the native Trident provenance model allows incomplete graphs (that are eventually consistent) to be present. Also, OPM graphs inserted into the OPM store from external sources are not copied to the Trident provenance store in this approach. So equivalent queries sent to both metadata stores can return different results.

Since query integration provides an OPM view on top of existing Trident provenance metadata, the metadata in both are consistent and timely. Any metadata populated in the Trident store is visible through the OPM view, though it might return an incomplete OPM graph. Use of OPM views limits the ability to import external OPM graphs into the OPM store; so there is no inconsistency due to external imported provenance and provenance collected in Trident.

The decoupled approach keeps the Trident and OPM provenance stores independent, and provenance graphs are exported to the OPM store from Trident native store on-demand. While there is consistency at the level of individual OPM graphs, the provenance stores themselves are not maintained in consistence. Also, provenance graphs are only exported from Trident to OPM and not the other way round. Therefore, provenance graphs from external systems imported into the OPM store will not be visible in the Trident store.

4.2.3 Storage Overhead and Performance

The approaches entail different overheads to collect and record provenance at runtime and for additional storage required. The performance overhead may be caused by the need to rationalize schema differences between OPM and native provenance collection at runtime, while the storage overhead may be due to duplicate copies of provenance metadata maintained. These factors affect scalability, both when the rate of provenance collection is high due to concurrent workflows and when the provenance collected increases over time.

The storage impact using the tight integration approach is minimal since the system stores just one copy of the provenance data using the OPM model, and the disk space for storing using the OPM model is no more than using Trident’s native model. There is, however, a performance penalty for mapping the runtime provenance events to OPM since the events may arrive out of order and existing metadata has to be examined before committing them to the OPM store to keep the OPM graphs consistent. Since the queries are triggered for each arriving event, the overhead can be high as the rate of events and size of OPM database increases.

In addition to the performance overhead for runtime event analysis for reordering, the storage integration approach also has increased storage overhead since there are two copies of the provenance metadata. The query integration approach has neither the storage space overhead nor a storage performance penalty since provenance is always recorded only in the Trident data model. The OPM database views do not affect provenance storage performance of the Trident native store.

The decoupled approach keeps two copies of the data, both in the Trident data model and the OPM model. However, duplicate copies are only created on-demand, when a user requests to export and import from either store into the other. There is no additional runtime performance impact for native provenance storage. The performance cost for import/export of provenance graphs is still less than runtime event analysis in the storage integration approach since the latter is more fine-grained and frequent compared to building whole OPM graphs for export.

4.2.4 Query Expressibility and Performance

The provenance queries that are feasible depend on three factors: the completeness and consistency of provenance information, the query features of the storage abstraction, and on the links that provenance has to related annotations on artifacts and processes. As example of the latter, native artifact identifiers used within Trident are set as the ‘value’ attribute of the artifacts in the OPM schema, allowing additional metadata about the artifact to be looked-up in the Trident Registry. If OPM and Trident stores use different storage abstractions, queries become complex, or worse, metadata mediation has to be done by the query client. The performance overhead of querying may also vary, depending on whether all metadata is collocated or storage boundaries have to be crossed. While this is less critical for one-off queries, the impact can be significant for systems that support large communities with frequent queries.

Semantics of the OPM and Trident models also plays a role in querying. If the native provenance model has semantically different attributes from the OPM specification, querying and even mapping during storage is a challenge. While this is not
an issue for Trident since its native model uses attributes analogous to OPM, other provenance systems that diverge in semantics may be affected.

The **tight integration** approach ensures consistency, but completeness may be lacking if all attributes present in the provenance events cannot be mapped to OPM. This is possible since the native Trident provenance schema is a superset of the OPM schema and, if OPM annotations are absent, this can cause metadata loss during mapping. However, rich query expressibility is ensured if the extra Trident metadata is retained separately with sufficient links to the OPM metadata (though interoperability degrades). Another consequence of using a single metadata model is good query performance since all metadata – provenance and related annotations – use the Trident Registry for storage and queries can run completely within a single storage backed of the Registry.

**Storage integration** has similar drawbacks as tight integration, except that the Trident native schema is guaranteed to retain metadata that do not map into OPM. Using independent stores for OPM and Trident native metadata impacts performance of queries that join data from both. This is mitigated by collocating both stores within the same Registry storage backend (e.g. SQL, CloudDB, etc.). Choosing different storage implementations can also constrain the variety of supported queries to the lowest common denominator of the two, which is possibly the .NET LINQ object queries used through the Registry APIs.

Views used by the **query integration** approach has consistency benefits similar to tight integration and metadata retention as seen in storage integration. While the use of SQL as the required storage backend limits users to queries that use SQL (or LINQ), it allows the full richness of SQL to be used uniformly. Views can inhibit query performance. Materializing the views to improve query performance brings with it associated storage overheads similar to storage integration. However, modern databases can alleviate the performance and storage penalties through careful schema modeling and use of indices.

The **decoupled** approach places the most restriction on queries since complete provenance is not present in either Trident or in the OPM stores to begin with. An explicit import or export operation is needed to load the provenance of interest before querying it. This import/export function may be automatically triggered once a complete provenance graph is available in either store. Other than this, performance and query expressibility are similar to the storage integration model.

### 4.2.5 Interoperability Features

The primary reason for using the OPM specification is to enable interoperability. While all the approaches support the OPM model to at least export Trident native provenance to the OPM model, their degree of interoperability with external provenance systems and provenance tools differs.

**Tight integration** allows refactored Trident provenance tools, such as the provenance browser and the provenance store, to be directly used by external groups to store, visualize, and query their OPM graphs. Similarly, Trident can potentially swap in external provenance tools in place of its built-in ones. This allows true interoperability with other OPM-based systems.

**Storage integration** and **decoupled** approaches allow the Trident Registry to be reused for standalone OPM metadata management. Likewise, these approaches can reuse other OPM storage systems instead of Trident Registry for storing the OPM metadata. Storage integration, in addition, allows external provenance browsers to be used to view the OPM store that contains both provenance metadata for all Trident workflows and provenance imported from external systems; the decoupled approach does not allow such browsing of Trident provenance unless it has been exported to the OPM store.

**Query integration** provides the least level of interoperability since the OPM “storage” is virtual – as views – that do not allow import of external OPM metadata, only export. Also, it limits itself to SQL as the sole data storage backend.

### 5 The Third Provenance Challenge

The third provenance challenge presented an opportunity for Trident to build compatibility with OPM and verify its interoperability against provenance collected from other teams. The **load workflow** used in the provenance challenge [8] was a variation of a workflow used in the Pan-STARRS project that was itself using the Trident Workbench [13]. So, it was trivial to (re)compose the challenge workflow in Trident and run it.

Based on the above analysis of approaches, we decided to adopt the decoupled approach for our implementation of OPM integration. One of the key reasons was to reduce coding and maintenance effort while improving interoperability features. Though this results in disconnecting the OPM document from additional workflow information present in the Registry, the ability to annotate the OPM entities – as better defined in the final OPM v1.1 specification post-challenge [11] – allows for suitable property-values to be encoded within the OPM document that act as links to more detailed native metadata in the Trident provenance and workflow models.
We used the XML schema for OPM from University of Southampton [14] as the representation of OPM documents exported and imported into the OPM store. Provenance collected for the load workflow in Trident was exported as OPM XML documents using our export utility and imported into the OPM store of the Trident Registry using the import utility (Figure 2(d)). Similarly, OPM XML documents from four other teams – COMAD from UC Davis, the entry from University of Southampton and ISI/University of Southern California, Taverna from University of Manchester, and Karma from Indiana University – were imported successfully into the OPM Registry and Core Provenance Queries 1 and 2 attempted on them. These entries are more fully described at their online Provenance Challenge websites [19] and the queries are described in the editorial accompanying this special issue [8]. All queries were run exclusively on the OPM store. Table 1 summarizes the results of the import and query tasks for the five teams, including Trident.

<table>
<thead>
<tr>
<th>Team</th>
<th>Import OPM</th>
<th>Query 1</th>
<th>Query 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>UC Davis/COMAD</td>
<td>Success</td>
<td><strong>Success.</strong> However, Detection ID value artifact in the exported OPM is incorrectly ‘GeneratedBy’ the LoadCSVFileIntoTable process instead of being ‘Used’ by it and needed correction. There is no database table artifact that can be the logical start of this query as collections are opaque.</td>
<td>Failed. Collections artifacts are opaque entities. It was not possible to establish a causal chain between the table name artifact (e.g. P2FrameMeta) and the IsMatchTableColumnRanges process.</td>
</tr>
<tr>
<td>University of Southampton/ISI</td>
<td>Success</td>
<td>Failed. The artifacts are opaque as their values are not embedded in the OPM XML or RDF. So it is not possible to map from the Detection ID to artifact.</td>
<td>Failed. Same reason as for Q1.</td>
</tr>
<tr>
<td>University of Manchester</td>
<td>Success</td>
<td><strong>Partial Success.</strong> Missing Date had to be added to DateTime.</td>
<td>Failed. Same reason as for Q1.</td>
</tr>
<tr>
<td>Karma3</td>
<td>Partial Success</td>
<td><strong>Success.</strong> However, only Detection ID values for artifact was present. So will not answer queries over other artifact fields.</td>
<td>Success.</td>
</tr>
<tr>
<td>Trident</td>
<td>Success</td>
<td>Failed. Trident does not record the tuple values for database tables as artifact, only their references. So the Detection ID value was missing in the provenance collected.</td>
<td>Success. Artifact simple value types were present and allowed answering this query.</td>
</tr>
</tbody>
</table>

As can be seen from Table 1, most teams generated valid OPM documents conforming to the OPM XML schema. Only Karma3 had an incorrect DateTime format with missing date value that was naively corrected using a default value.

Trident’s decoupled approach to OPM integration, while having lower coding overhead, caused it to fail to answer Query 1 which depends on artifact’s actual value. The artifacts in the load workflow were database tuples rather than simple types, and Trident activities do not support serialization of artifacts that refer to databases. The limitation is with the provenance collection in Trident rather than the OPM Store that uses the Trident Registry. As can be seen for the results for the other teams, the Registry is able to answer this query for provenance imported from other teams. This highlights the advantage of interoperable provenance tools, where a tool from one team can be reused (even more effectively) by other groups. The way in which provenance itself is mapped from native provenance models to OPM and the associated loss of semantics is often the constraining factor.

The result of running the queries highlights some issues in the approaches taken by teams to export OPM:

- Publishing only opaque references to the artifacts in the OPM graphs prevents the ability to answer queries based on artifact values. Some teams had purely opaque artifact references that cannot be interpreted without knowing the addressing and storage scheme. For example, UManchester use URIs for artifacts that do not reveal anything about the artifact value itself. Some teams like Karma embedded partial artifact values, such as ‘Detection ID’, in the OPM document. COMAD published complete artifact values in the OPM graph, helping answer the query 1 (but consequently bloating the OPM graph size). Better specification of annotations, such as using profiles for artifact and process dereferencing to access associated metadata, can help overcome this issue.
- Collections of artifacts were opaque in COMAD and could not be interpreted. Suggestions for an OPM collection profile that has been recently forthcoming [16]. This would allow such compound artifacts to be sufficiently decomposed as part of the OPM model to discern the provenance of individual items rather than using implementation specific semantics.
• The degree of detail published in the OPM graphs varied between teams. While export and import of OPM may be information neutral for the simple core queries, there is possibility for information loss for more complex ones. Future challenges may try to address this issue.

6 Conclusion

The provenance challenge helped to bring OPM compatibility into the Trident Workbench and served as an opportunity to study different approaches to support OPM in existing workflow and provenance systems. These diverse approaches vary in programming ease, performance penalties and interoperability features gained. Consistency and information loss of native provenance and OPM were also examined. The analysis can help existing developers and practitioners of provenance systems to determine the approach best suited for them to gain compatibility with OPM for interoperability.

The OPM documents published by other teams provided an indication of the possible diversity in provenance export and the potential pitfalls despite having a core specification. Some of the problems that arose during the challenge have been addressed in the latest Open Provenance Model v1.1 specification while other are being examined. We expect to use this experience to better integrate OPM into Trident and attempt subsequent challenges using the latest specification.

References


