PBML

The Prague Bulletin of Mathematical Linguistics

NUMBER 100 OCTOBER 2013 91–100

Open Machine Translation Core: An Open API for Machine Translation Systems

Ian Johnson

Capita Translation and Interpreting

Abstract

Open Machine Translation Core (OMTC) is a proposed open API that defines an application programming interface (API) for machine translation (MT) systems. The API defined is a service interface which can be used to underpin any type of MT application. It consists of components which allow programmers, with little effort, to integrate different MT back-ends into their applications since an OMTC compliant MT system presents a consistent interface. OMTC attempts to standardise the following aspects of an MT system: **resources** – the abstract representation of assets used e.g. documents and translation memories, **sessions** – a period of time in which a user interacts with the system, **session negotiation** – agreement on which services are to be provided, **authorisation** – integration with third party authorisation systems to prevent users performing unauthorised actions, **scheduling** – the management of long running MT tasks, **machine translation engines** – a representation of an entity capable of providing only MT, and **translators** – a conglomeration of, at least one of the following, an MT engine, a collection of translation memories, and a collection of glossaries.

1. Introduction

Open Machine Translation Core (OMTC) is a proposed and open API for the construction of machine translation (MT) systems (Johnson, 2013). The central idea of OMTC is to be able to easily integrate disparate back-end MT systems together into an application such that the back-ends “look” consistent no matter the flavour of MT.

To identify the aspects and concerns that would be common to MT systems a use case analysis was carried out. Once the *actors* and use cases were catalogued then use cases which *any* MT system would require were identified. This reduced set was expanded into UML class diagrams to define the abstract OMTC specification. How-

Corresponding author: ian.johnson@capita-ti.com
Cite as: Ian Johnson. Open Machine Translation Core: An Open API for Machine Translation Systems. The Prague Bulletin of Mathematical Linguistics No. 100, 2013, pp. 91–100. doi: 10.2478/pralin-2013-0015.

ever, OMTC does define concrete classes where necessary. This paper gives a fairly high level description of the OMTC specification with a view that the reader study the full specification for details. OMTC attempts to standardise:

- **Resources**: the abstract representation of assets used by users in an MT system, e.g. documents and translation memories,
- **Sessions**: a period of time in which a user interacts with the system, e.g the time between login and logout,
- **Session Negotiation**: agreement on which services are to be provided,
- **Authorisation**: integration with third party authorisation systems to prevent users performing unauthorised actions,
- **Scheduling**: the management of long running and computationally expensive MT tasks,
- **Machine Translation Engines**: a representation of an entity capable of providing only MT, and
- **Translators**: a conglomeration of, at least one of the following, an MT engine, a collection of translation memories, and a collection of glossaries.

Figure 1 shows an example of how OMTC could be implemented. The figure shows two example applications: a client-server and a command line application. OMTC sits low down in the stack. OMTC's position gives the application programmer much more flexibility and freedom to use technologies and networking protocols that are available to them. For example, TAUS published their open MT system API which is designed to work as a RESTful web-service over HTTP (TAUS, 2012). Implementers of this API are tied to using HTTP. Using HTTP may not be desirable in some customer deployments, for example messages queues may have to be used. OMTC, on the other hand, is not tied to any technology and is reusable since it concentrates on one aspect of an MT system: *machine translation*. Moreover, the TAUS API specifies which methods are available to consumers of their service. If methods or arguments are required to be augmented the implemented MT system becomes non-compliant. OMTC allows the implementer to specify the methods and arguments required for their MT system.

Below OMTC sit the translation providers. Figure 1 shows the following disparate MT systems:

- **SmartMATE**: a self-serve SMT system allows API calls, via its RESTful web-interface, to build translation engines and start translations (Way et al., 2011).
- **Moses**: an open-source suite of tools for SMT engine development and translation. Integrating to an OMTC system would probably take the form of wrapping the existing command-line tools (Koehn et al., 2007).
- **SYSTRAN**: A rule-base MT system with an API available in their *Enterprise Server* product (SYSTRAN , 2008).
- **SDL Trados**: A computer-aided translation suite which presents an API called *SDL OpenExchange* (`http://www.sdl.com/products/sdl-trados-studio/`).

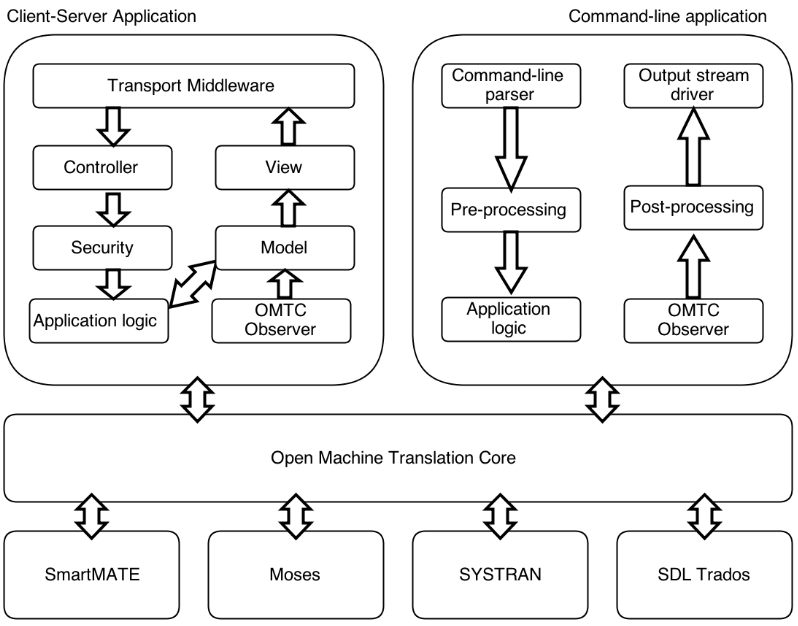

Figure 1. Example OMTC compliant applications

OMTC attempts to make these proprietary APIs homogeneous by defining an abstract interface for machine translation tasks and maintenance.

Further to the abstract specification, a reference implementation has been constructed using Java v1.7. It is released under a LGPL v3 license and is available by cloning the GitHub repository `https://github.com/ianj-als/omtc.git`. This implementation was written to provide an implementation that could be immediately used by developers to write OMTC compliant applications. The OMTC reference implementation is being used, at Capita Translation and Interpreting, to re-factor the SmartMATE application.

There follows a brief description of the common *actors* that would use an MT system. These *actors* were the central basis on which OMTC was designed.

2. Actors

An actor specifies a role played by a "user" that interacts with a system, but is not a part of that system. Actors are external to the system and can represent a human, external software, or an external system etc. (Obj, 2007).

There are three principal actors in an MT system:

- **Translator**: This actor's role is to perform translations and is the main end-user of an MT service. All other actors provide means to provide resources so that the translator may schedule translation tasks. Since this actor is expected to be widespread it attracts the fewest number of possible actions in the MT service and those actions are primarily read-only. Therefore, the scope to which this actor can intentionally harm the MT service is kept to a minimum. Moreover, this actor requires very little knowledge of MT in order to complete translation tasks.
- **Engine Manager**: This actor is able to mutate MT engines. The primary role of this actor is to maintain MT engines, e.g., train, re-train, compose or destroy MT engines. This actor should have a reasonable understanding of MT and the kinds of MT that the implementation is supporting. For example, if the implementation supports SMT then this actor would have an understanding of how to take a tabula rasa system and build an engine for use by the translator actor. Moreover, this actor is responsible for determining who is able to use the engines which the actor constructs. The use cases available to the engine manager actor is the union of those use cases for this and the translator actor.
- **Administrator**: The administrator actor is permitted to manage users for a particular customer. Customers may have many users which are translators or engine managers. Managing which use cases a customer's users are permitted to perform is the administrator actor's remit. This actor would be authorised to choose the payment plan, if one is required, and make payments for the use of the MT service. The administrator actor is permitted to invoke the use cases available to engine manager and translator actors.

Considering each of these actors, a number of *concerns* have been arrived at that are believed to be common to many MT systems. The concerns are collections of use cases and are described below.

3. Resource Management

A *resource* is an object that is provided or constructed by a user action for use in an MT system. A non-exhaustive list of examples is: document files, translation memories, glossaries, or MT engines. *Resource management* is a collection of use cases that allow all actors to load, construct, catalogue, and remove resources from an MT system. For example, if the MT system were a web-service then making a translation memory available to the MT system would probably be an upload action.

Resources may need some kind of ownership. If an MT system is a standalone command line driven application this may not be necessary, or the file-system can provide this feature: read or write permissions on files being used by the running process will be determined by the user running the process. However, if an MT system is a multi-user service then ownership of resources would become necessary. Users from different customers should not be permitted to access *any* resource constructed or made available to the system by other customer users.

OMTC defines two kinds of resource:

1. **Primary resources**: any resource that has been constructed externally and made available, in some way, for use in an MT system. Examples of these resources are: a document, a translation memory (TM), a glossary etc. If these resources are required for future use it is recommended that these resources be persisted. Primary resources are immutable, i.e. if a resource's content is to be altered it is a distinct resource.
2. **Derived resources**: these resources are constructed using their primary counterparts either as a conglomeration or a separate entity is created, e.g. creating a SMT engine using a translation memory (a primary resource) to create a derived resource: the engine itself.

4. Sessions, Negotiation and Authorisation

In order for users to be able to use an MT service the API needs an idea of a *session*. A session is the period in which a user will interact with an MT service. An MT application may need to acquire the identity of users, whilst other implementations may not. Therefore, the OMTC API needs to support both user identity and anonymity. Moreover, clients to an MT service will support certain exchange formats, and expect certain features from the application. A *session negotiation* is defined in the API in order that both client and server can ascertain if, once the session is set up, their expectations of each other is correct. If a user's identity is to be determined then the application can restrict the actions a user can perform based on their role(s), i.e. *authorisation*. OMTC models these aspects.

4.1. Sessions

A session is a period in which a user interacts with an MT system. OMTC places no restrictions an application's definition on a session other than this. Sessions could be defined by the time between login and logout, the lifetime of a console application, or persisted over many login/logouts. Sessions can be associated with a user where user identity is necessary, for example in a pay-as-you-go web-application. In applications where user identity is not required an OMTC session supports not being associated with a user. An example of this type of application would be a console command line application where the user is explicit: the user running the program. All actions in an OMTC application are done on behave of a session.

4.2. Session Negotiation

An optional part of the OMTC specification is *session negotiation*. Session negotiation is a protocol which allows the provider and consumer of an MT service to come to some agreement on what can be expected from the provider. If session negotiation is implemented clients, including other MT systems, can discover which features are supported, and which requirements are necessary. The features and requirements are modelled as *capabilities*. Capabilities come in four flavours:

- **API**: This capability, today, only specifies the version of the API being used.
- **Resources**: These capabilities describe the file types that the service can support. Supporting means that the service will store and use the resource in an appropriate way.
- **Features**: The actions that can be expected from an MT service, but may not be available in every MT service.
- **Prerequisites**: The prerequisites that client shall ensure are true before some or all of the MT service's features become unavailable to a client, e.g. payment.

During negotiation the unsupported capabilities are returned to the consumer. If provider has determined that the consumer cannot have a meaningful conversation then the session is closed. However, the consumer can close the session if it receives unsupported capabilities on which it depends. Session negotiation must be completed before the consumer completes session initialisation.

4.3. Authorisation

OMTC does not specify *any* security features. It is the application's responsibility to integrate with authentication systems. However, if authorisation is required in an MT system then some integration with the external authentication provider is necessary to provide user identity, and authorisations. The specification provides two interfaces to interlock an external authentication provider.

5. Scheduling

Machine translation consists of a number of operations which are computationally expensive. Constructing an MT service with many users requires that the computational resources are shared *fairly* between the demands of the users. The implementer of an MT service needs to define:

- Which computational resource or resources will be used to execute the computationally expensive operations,
- The latency of an operation before it is executed, and
- A policy to determine how users' operations will be scheduled, i.e. priority.

The scheduling API, defined by OMTC, needs to support different kinds of computation resource management: from native threading to distributed resource management products. The pattern used in the scheduling API is detached execution with notification on completion, whether successful or not.

5.1. Tickets

The scheduling API issues *tickets* when an operation is submitted to the underlying detached execution implementation. A ticket is a receipt for, and uniquely identifies an operation. When the operation is submitted an *observer* will be provided which observes the progress of the computation. On completion, the observer is invoked with the appropriate ticket to identify which operation has completed. This is the observer design pattern (see Gamma et al., 1994). The observer is application defined and is used to update any data that relies on the computation.

Operation priorities are defined using the scheduling API. This allows an application defined priority to be used to prioritise operations into the particular detached execution environment. For example, a priority could, say, for a paid-for MT service prioritise operations, invoked by users, which are on a higher tariff. So, say, a user on a *Freemium* tariff would have their operations prioritised lower than a user who pays for the service. Depending on the detached execution environment a priority might determine, not only, the latency of an operation, but also how much processor time a certain operation can expect when being execute.

6. Machine Translation Engines

A *machine translation engine* is defined as an entity that will solely perform machine translation. This may be a decoding pipeline in an SMT system or software that implements a rule based system. MT engines are built using primary resources and generally use computationally expensive operations to produce translations. Engines shall have operations available that, depending on their nature, shall read or mutate engine state, e.g.

- Evaluating an engine,
- Composing engines,
- Testing engines, and
- Training SMT engines.

Mixin interfaces are used to add optional functionality to an MT engine. This allows the application programmer to choose mixins useful to the kind of MT engine being implemented. Using mixins in this way prevents the application programmer from being tied to this API; it does not mandate that any class inheritance is used. This is particularly useful when using languages that do not support multiple inheritance and can be used alongside existing frameworks and class hierarchies.

The mixins provided define the following operations:

- **Composition**: compose one MT engine with another,
- **Evaluation**: score an MT engine,
- **Update parameters**: mutate runtime options/parameters,
- **Querying**: invoking translations one sentence at a time,

- **Training and retraining**: specifically for SMT engines to build appropriate models,
- **Testing**: provide resources to test a constructed MT engine, and
- **Updating**: mutation of an existing engine to adapt to new data or rules.

The operations, in the mixins, that could represent computationally expensive operations and use an asynchronous invocation pattern. In order to track the operation the caller of these methods receives a *ticket*. The ticket is used to represent an "in flight" operation and, once complete, will be used in a notification. Notifications are used to inform the application of the state of a completed operation: submitted, starting, or completed successfully or failed.

7. Translators

Translators are a conglomeration of an MT engine, translation memories and glossaries. A translator will specify at least one of these resources. This allows translators to support translations using any combination of MT, TM or glossaries. It is the responsibility of application programmers to handle these resources in an appropriate way for the flavour of translation required.

Translations are typically computationally expensive and can take a considerable amount of time to complete. In an MT system that is multi-user computation resources should be shared fairly between the demands of the submitted translations. As with MT engine operations, translations shall be ticketed and a ticket observer is required to receive notifications of the progress of a translation task.

There are two methods of performing a translation:

- **Primary Resource Translation**: A primary resource made available to an MT system can be translated. It is application defined as to which kind of primary resources are supported for translation. If supported, it is implementation defined as to whether any pre- or port-processing is required, e.g. file filtering.
- **Sentence-by-sentence Translation**: Translations can be supported that consist of a single sentence. MT engines can be queried sentence-by-sentence to perform a translation using only the engine. However here, TM and glossaries can be mixed into a richer translation that uses any translation pipeline that may be implemented.

8. Language Bindings

The OMTC specification is documented using UML which is a language-agnostic representation. It is expected that any modern computing language is capable of implementing the OMTC specification. OMTC defines some *generalised classes*. Generalised classes are classes which require types arguments to construct the class. Concrete implementations of this is are Java and C# generics, and C++ templates. Many, if not all, of the extant non-object oriented computing languages are not capable of implementing these classes. However, the solution is to design an OMTC implementation that builds concrete representations of the OMTC generalised classes. Functional programming languages are also candidates for use in implementations. Haskell's typeclass language feature would be particularly suited to an OMTC implementation.

The OMTC specification comes with a Java v1.7 reference implementation. This implementation was constructed to allow people to view the specification in code to gain deeper understanding, and, if they wish, to build their own OMTC compliant MT system with Java.

Implementations in other languages are encouraged. With the popularity of web-frameworks, such as Spring MVC (Yates et al., 2013), Rails (Hart, 2012) and Django (Alchin, 2013), Ruby and Python implementations are welcome since they'll provide an easy way to build web-hosted MT services.

9. Summary

A proposed open API for MT systems, called Open Machine Translation Core, has been presented. It attempts to standardise common aspects and concerns to all MT systems. It is believed that this abstract interface will underpin and ease the development of *any* MT system being developed. Whilst this is only a high level view of the proposed API it is recommended that the reader view the full and entire specification. The full specification and a Java reference implementation is freely available, under a LGPL v3 license, from GitHub by cloning `https://github.com/ianj-als/omtc.git`.

Acknowledgements

This work was done as part of the MosesCore project sponsored by the European Commission's Seventh Framework Programme (Grant Number 288487).

Bibliography

Alchin, Marty. *Pro Django*. Apress, 2nd edition, 2013.

Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1st edition, 1994.

Hart, Michael. *Ruby on Rails Tutorial: Learn Web Development with Rails*. Addison Wesley, 2nd edition, 2012.

Johnson, Ian. *OMTC: Open Machine Translation Core*, Version 0.6.1-DRAFT edition, 2013. URL `https://github.com/ianj-als/omtc/blob/master/documentation/omtc.v0.6.1-DRAFT.pdf`.

Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, ACL '07, pages 177–180, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics.

OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2. Object Management Group, Inc., 2007. URL `http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF`.

SYSTRAN (2008). *SYSTRAN Enterprise Server 6: API Reference Guide.* SYSTRAN, 2008. URL `http://www.systransoft.com/download/user-guides/SYSTRAN.ses6-api-reference-guide.pdf`.

TAUS(2012). *A Common Translation Services API.* TAUS, September 2012. URL `https://labs.taus.net/interoperability/taus-translation-api`.

Way, Andy, Kenny Holden, Lee Ball, and Gavin Wheeldon. SmartMATE: Online self-serve access to state-of-the-art SMT. In *Proceedings of the Third Joint EM+/CNGL Workshop "Bringing MT to the User: Research Meets Translators"*, pages 43–52, 2011.

Yates, Colin, Seth Ladd, Marten Deinum, Koen Serneels, and Christophe Vanfleteren. *Pro Spring MVC: With Web Flow.* Apress, 2nd edition, 2013.

Address for correspondence:
Ian Johnson
`ian.johnson@capita-ti.com`
Capita Translation and Interpreting
Riverside Court, Huddersfield Road
Delph, Lancashire
OL3 5FZ, United Kingdom