4.1.4 Refactoring

The solution of the optimization problem produces a graph representation of the microservice architecture, in which methods and domain entity classes (i.e., nodes) are grouped into microservices. The implementation of the identified microservices can be automatically realized by suitably "moving" methods and classes into the right microservice – hence reusing the monolith code – and generating the new API controllers needed to realize the (new) inter-microservice communication. However, it is important to remark that this automated process can be actually performed only if the code of the monolith is suitable to be reused; if, for any reason, this can not be done, the skeleton code of the methods and API interfaces can be still generated. These concerns are discussed in Section 4.3.

The refactoring step generates the implementation of the identified microservices by (i) "moving" the code of each method into the right microservice; (ii) adding new methods for exposing and consuming APIs (*API controller synthesis* and *API consumer synthesis*); (iii) placing (and replicating) the domain entity classes into all the microservices requiring them.

In the following, we describe the rules that allow the generation of API controllers and API consumer methods and the replication of domain entities.

Rule 1: API Controller synthesis

An API controller method is generated for each method that is called from at least one different microservice. That is, in each microservice M_k, an API controller method i' is generated for each method $i \in M_k$ s.t. exists an arc $(j,i)|j \notin M_k$. The generated API controller will receive the API calls from outside the microservice and will invoke the method i. If needed, the methods in the microservice M_k will invoke the method i without using the API.

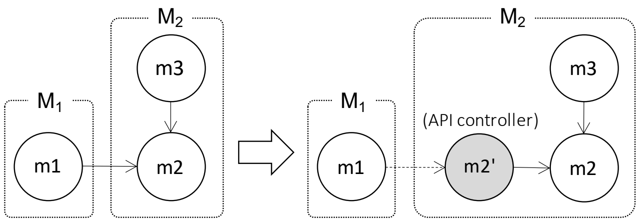


Figure 4.6: Example of API controller synthesis

Figure 4.6 shows an example of the application of this rule: the node representing the method $m2$ in the microservice M_2 has an ingoing edge from $m1$, which belongs to a different microservice (left-hand side). The method $m2'$ is generated as an API controller for $m2$ (right-hand side). It allows exposing the API to the outside of the microservice and will be used to invoke the method $m2$.

Rule 2: API Consumer synthesis

An API consumer method is generated to call methods that have been placed into different microservices. For this reason, in each microservice M_k, a method j' is generated for each arc (i,j) s.t. $i \in M_k$ and $j \notin M_k$. The new method contains the code needed to invoke the API related to the method j.

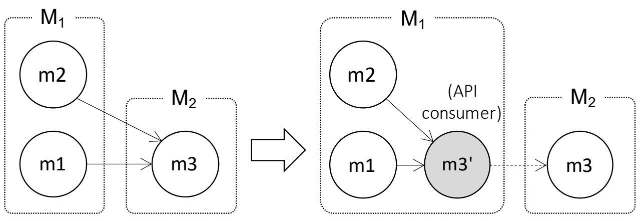
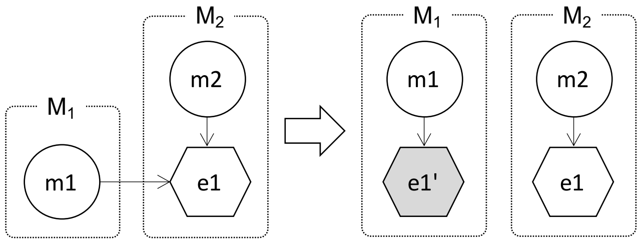


Figure 4.7: Example of API consumer syntesis

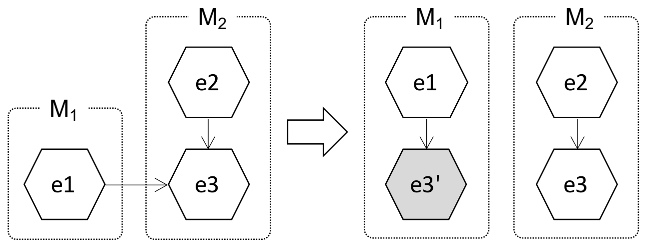
Figure 4.7 shows an example of the application of this rule. The nodes representing the methods $m1$ and $m2$ in the microservice M_1 have an outgoing edge towards $m3$, which belongs to a different microservice (left-hand side). The method $m3'$ is generated in M_1 as an API consumer for $m3$ (right-hand side). It allows consuming the exposed API for j (as described in Rule 1) and, hence, invoking the method j.

Rule 3: Entities replication

Alongside the API controllers and consumers, methods must have the access to all the required entity classes. Thus, all the entity classes that are referenced by a method are copied into the microservice that hosts the method, together with all the other entities connected to the first entity class. Moreover, also entities that are referenced by other entities have to be copied into the same microservice. In other words, for each microservice M_k, an entity class e is put into the microservice M_k iff there exists a method-to-entity or an entity-to-entity edge (i, e) s.t. $i \in M_k$.



(a) Case A



(b) Case B

Figure 4.8: Example of entities duplication

Figure 4.8 shows the application of the rule. It considers two cases. In the first case (Figure 4.8a), the entity $e1$ in M_2 is "used" by the method $m1$, which is in the microservice M_1 (left-hand side). The entity is replicated into the microservice M_1 (right-hand side). In the second case (Figure 4.8b), the entity $e3$ in M_2 is referenced by the entity $e1$, which is in the microservice M_1 (left-hand side). The entity is replicated into the microservice M_1 (right-hand side). This refactoring rule is applied iteratively until there are no dependencies between entities spanning across different microservices.