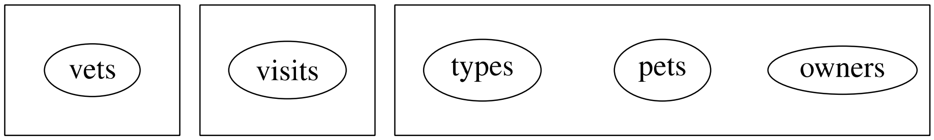

Figure 6. Optimization problem output

```java
@Service
public class ClinicServiceImpl implements ClinicService {
   // code ommitted for brevity
   private VisitRepository visitRepository;
   // code ommitted for brevity
   @Override
   @Transactional
   public void saveVisit(Visit visit) throws
         DataAccessException {
      visitRepository.save(visit);
   }
}
```

```java
@Repository
public class JpaVisitRepositoryImpl implements
      VisitRepository {
   // code ommitted for brevity
   @Override
   public void save(Visit visit) {
      if (visit.getId() == null) {
         this.em.persist(visit);
      } else {
         this.em.merge(visit);
      }
   }
}
```

```sql
void upsert_visit(visit){
   if visit.id IS NULL then
      INSERT INTO
               visits
      VALUES   (visit.date,
               visit.description,
               visit.pet_id);
      return;
   end if;

   SELECT id
   FROM visits
   INTO v_id
   WHERE visit.id = id;

   if v_id IS NULL then
      abort;
   end if;

   UPDATE visits
   SET date = visit.date,
      description = visit.description,
      pet_id = visit.pet_id
   WHERE visit.id = id;
}
```

Figure 7. Application logic (left) extracted to a reactor function (right)

6. Discussion and Limitations

This study proposes an exact optimal approach for allocating relational tables, REST interfaces, and application logic to clusters, which could represent services or reactors. To the best of our knowledge, there is no identical work in literature. Even though our system formalization is based on the work of Levcovitz et al. [Levcovitz et al. 2016], it goes much beyond in both presenting a MIP-solver-based automatic method for distribution among clusters as well as considering heuristics to identify application logic in source code to be extracted.

One of the limitations of our technique concerns the assumption that the system adopts a three-tier layered REST-based architecture. However, it represents a widely adopted architecture in industrial settings. Also, while our technique currently only considers interfaces that enable GET and POST operations, we don't see significant constraints to extend it with DELETE and PUT operations.

Regarding the evaluation, the prepared artificial workload may not provide sufficient coverage for all cases in which the technique could be applied. Nevertheless, the workload was defined based on reasoning about the application domain. It is noteworthy to mention that the workload, its limits and the source code metrics were verified by three independent researchers. Additionally, to test the sensitivity of our model, we have also applied it to a different hypothetical workload, allowing to observe sensitivity of model output due to changes in the input specifications.

Finally, we chose a specific Java software project for applying our technique. However, we believe the technique is generic enough to be applied to other object-oriented programming languages (e.g., C#) and frameworks (e.g., .NET Core).