An Extensible, Model-Driven and End-User Centric Approach for API Building

José Matías Rivero[1,2], Sebastian Heil[3], Julián Grigera[1], Esteban Robles Luna[1], and Martin Gaedke[3]

[1] LIFIA, Facultad de Informática, UNLP, La Plata, Argentina
{mrivero,julian.grigera,esteban.robles}@lifia.info.unlp.edu.ar
[2] Also at Conicet
[3] Department of Computer Science, Chemnitz University of Technology, Germany
{sebastian.heil,martin.gaedke}@informatik.tu-chemnitz.de

Abstract. The implementation of APIs in new applications is becoming a mandatory requirement due to the increasing use of cloud-based solutions, the necessity of integration with ubiquitous applications (like Facebook or Twitter) and the need to facilitate multi-platform support from scratch in the development. However, there is still no theoretically sound process for defining APIs (starting from end-user requirements) or their productive development and evolution, which represents a complex task. Moreover, high-level solutions intended to boost productivity of API development (usually based on Model-Driven Development methodologies) are often difficult to adapt to specific use cases and requirements. In this paper we propose a methodology that allows capturing requirements related to APIs using end-user-friendly artifacts. These artifacts allow quickly generating a first running version of the API with a specific architecture, which facilitates introducing refinements in it through direct coding, as is commonly accomplished in code-based Agile processes.

Keywords: API, Model-Driven Development, Agile Development, Prototyping.

1 Introduction

Over the last years, users and businesses have witnessed a trend to *move* applications and services to the cloud. Several aspects motivate this trend, most importantly cost and deployment time. In this context, developers must interact with applications and services they do not directly control, so they need APIs to facilitate this interaction. Since APIs generally centralize operations and business-logic among applications in several platforms, they are inherently complex; however, most development processes do not take this into account, particularly Agile methodologies, which do not provide clear and structure method to cope with the complexity of API design [1]. To tackle such complex requirements, we have introduced a Model-Driven Development (MDD) [2] solution called *MockAPI* [1], which is limited to providing a prototypical version of the API. In this paper we propose *ELECTRA (*standing for ***E****xtensible mod-e****L****driven* ***E****nduser* ***C****en****TR****ic* ***A****PI)*, an hybrid Agile, MDD and coding approach that

(1) uses an end-user friendly language to define API-related requirements as in MockAPI (annotated mockups), (2) allows to quickly define and generate a running API for testing integration with other software artifacts, and (3) proposes an API runtime that can be extended (hence the term *Extensible*) with custom code without breaking the model's abstraction. We chose mockups as our main requirement artifact because of their positive results in agile approaches [3], and their valuable requirements communication capabilities [4]. Using mockup and annotations as an end-user friendly language we intend to capture the complex API requirements and, at the same, time, provide a framework for quick API generation and refining.

2 The ELECTRA Approach

The ELECTRA process, depicted in Figure 1, is an adaptation of Scrum [5], the most widely used agile process in industry [6]. As in Scrum, every iteration in the ELECTRA approach starts by selecting the User Stories to be tackled (*Define Sprint Backlog* step). Then, ELECTRA mandatorily requires building mockups with essential end-user participation to concretize each of these stories (*Mockup Construction* step). After all User Stories are associated to mockups, developers use an enhanced version of the MockAPI tool [1] to tag the mockups with API-related annotations (*Mockup Tagging* step in Figure 1). These annotations are based on a simple grammar that makes them easy to understand by end-users. From the annotated mockups, an API can be derived through a code generation process (*API Generation* step). At this point, developers get a running usable API for integration testing with other software artifacts (*API testing* step). The generated API is deployed to ELECTRA's runtime environment which allows developers to refine it through direct coding (*API Refining*). The result of this process is the Final API for the current iteration (*API Increment* step).

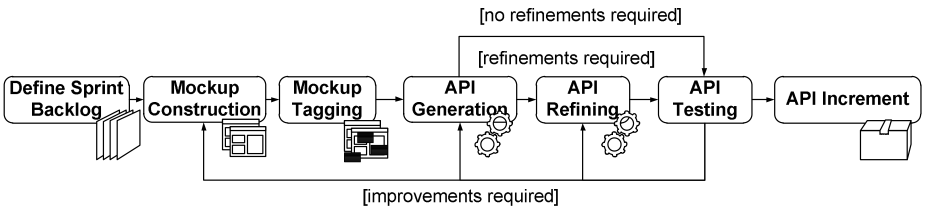
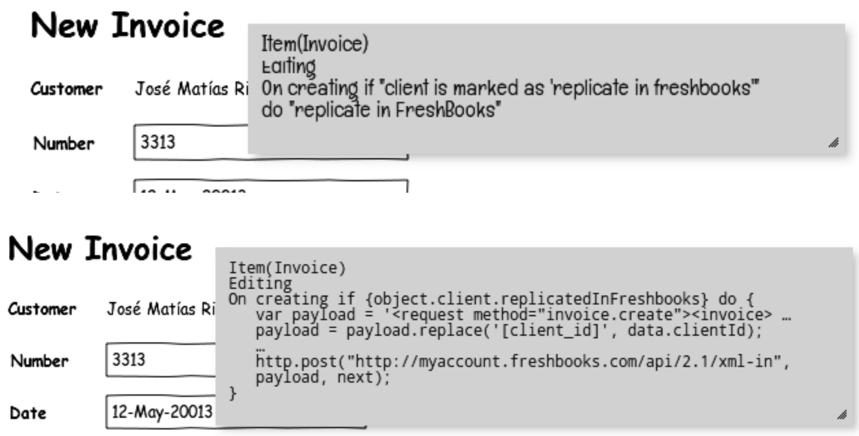


Fig. 1. The ELECTRA workflow

Any type of user interface mockup can be used with ELECTRA tooling. After mockups have been imported, different kinds of annotations can be defined by the engineers in presence and with collaboration of end-users. The three most important annotations types are *Data annotations*, which allow defining object types or business entities well-known by stakeholders, *Constraint annotations*, which enable the definition of business rules and *Action annotations*, which describe the execution of heterogeneous or complex tasks within the API. Besides its technical specifications (understood by engineers), some annotations provide an end-user friendly structured text mode to describe API requirements in natural language. While end-users require engineers help to write annotations, this text mode eases their understanding. ELECTRA tooling currently uses JavaScript as its default scripting language, but any scripting language implementing the Java Scripting API can be used instead. In Figure 2 a *Data* and *Action* annotations are shown applied over an invoice management application to specify the existence of an Invoice business object and how it must be integrated with an external API.



```
Item(Invoice)
Editing
On creating if "client is marked as 'replicate in freshbooks"
do "replicate in FreshBooks"
```

```
Item(Invoice)
Editing
On creating if {object.client.replicatedInFreshbooks} do {
   var payload = '<request method="invoice.create"><invoice> …
   payload = payload.replace('[client_id]', data.clientId);
   …
   http.post("http://myaccount.freshbooks.com/api/2.1/xml-in",
   payload, next);
}
```

Fig. 2. Annotations in its different modes: end-user friendly (upper) and developer (lower)

3 Architecture and Code Generation

The core of the ELECTRA tool implementation (the so-called ELECTRA API runtime) defines and implements a RESTful API as a set of Endpoints, which are composed by an HTTP method, a URL regular expression and a script. When a new HTTP request is received by the runtime, it seeks for a matching Endpoint (with the same HTTP method and a matching URL regular expression). If it finds one, it executes its internal script. Endpoint scripts can access and invoke other *Scripts*, use stored *Resources* (for instance, media content as images or video) or invoke operations on *Services,* which are declared and customized by developers.

ELECTRA's code generation process consists in the generation of a set of Endpoints and Scripts, and the configuration of a default DB Service for storage purposes analyzing the annotated mockups. After triggering a code generation, developers can tune the API as much as needed just editing the required Scripts and Endpoints or changing the DB Service used. Also, they can define new Resources, Services or Endpoints manually. Scripts are generated respecting the Pipes and Filters pattern (where every Filter is formed by a Script implementing a different concern – i.e., code for a different annotation type –, thus facilitating and isolating changes in the API.

When triggering a code regeneration, edited Scripts (Filters) are not altered, thus preserving changes incorporated through direct coding.

4 Related Work

Benefits of using mockups as a primary requirement artifact in the development process have been reported through statistical studies [7]. Also, its benefits in the context of Agile and MDD processes (even API generation) have been commented [8]. In addition, in our previous work [8] we demonstrated that mockup and annotations provide a modeling framework that results more efficient than manual modeling, even considering conceptual models – which are very similar to the *Data annotations* presented in this work. However, none of these works propose a Model-Driven approach that can be extended through direct coding even in the models, as ELECTRA provides. Several MDD approaches specifically tackle RESTful APIs [9,10] (as ELECTRA), but they not provide a clear and pattern-based codebase allowing the quick introduction of detailed implementation as ELECTRA does.

References

[1] Rivero, J.M., Heil, S., Grigera, J., Gaedke, M., Rossi, G.: MockAPI: An Agile Approach Supporting API-first Web Application Development. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 7–21. Springer, Heidelberg (2013)

[2] Kelly, S., Tolvanen, J.-P.: Domain-Specific Modeling: Enabling Full Code Generation. Wiley-IEEE Computer Society (2008)

[3] Ferreira, J., Noble, J., Biddle, R.: Agile Development Iterations and UI Design. In: Agil. 2007 Conf., pp. 50–58. IEEE Computer Society, Washington, DC (2007)

[4] Mukasa, K.S., Kaindl, H.: An Integration of Requirements and User Interface Specifications. In: 6th IEEE Int. Requir. Eng. Conf., pp. 327–328. IEEE Computer Society, Barcelona (2008)

[5] Sutherland, J., Schwaber, K.: The Scrum Papers: Nuts, Bolts, and Origins of an Agile Process

[6] VersionOne Inc., State of Agile Survey (2011)

[7] Ricca, F., Scanniello, G., Torchiano, M., Reggio, G., Astesiano, E.: On the effectiveness of screen mockups in requirements engineering. In: 2010 ACM-IEEE Int. Symp. Empir. Softw. Eng. Meas. ACM Press, New York (2010)

[8] Rivero, J.M., Grigera, J., Rossi, G., Luna, E.R., Montero, F., Gaedke, M.: Mockup-Driven Development: Providing agile support for Model-Driven Web Engineering. Inf. Softw. Technol., 1–18 (2014)

[9] Pérez, S., Durao, F., Meliá, S., Dolog, P., Díaz, O.: RESTful, Resource-Oriented Architectures: A Model-Driven Approach, in: Web Inf. In: Chen, L., Triantafillou, P., Suel, T. (eds.) WISE 2010. LNCS, vol. 6488, pp. 282–294. Springer, Heidelberg (2010)

[10] Valverde, F., Pastor, O.: Dealing with REST Services in Model-driven Web Engineering Methods. In: V Jornadas Científico-Técnicas En Serv. Web y SOA, JSWEB (2009)