Kronos : A DSL for scheduled tasks based on textX

Miloš Simić, Novak Boškov, Aleksandar Kaplar and Igor Dejanović
Faculty of Technical Sciences, Novi Sad, Serbia
milos.simic@uns.ac.rs

***Abstract*—Scheduled tasks are a ubiquitous part of our daily lives, whether generating reports for monthly data analytic or sending newsletters to subscribed customers is needed. Domain-Specific Languages (DSL) are a viable approach that promise to solve a problem of target platform diversity as well as to facilitate rapid application development and shorter time-tomarket. This paper presents Kronos, a cross-platform DSL for scheduled tasking implemented using textX meta-language. Tasks described using Kronos DSL can be automatically created and started with provided task-specific information**

Keywords: DSL, Scheduled tasks, Python

I. INTRODUCTION

Kronos [1] is a Domain-Specific Language (DSL) for specification and execution scheduled tasks. Domain-Specific Languages are languages focused on a particular domain [2, 3]. Conversely, General-Purpose Languages (GPL) are languages that are made to solve as many problems as possible regardless of the domain (examples of GPLs are Java, C, Python). Certain studies show that DSLs can increase productivity by up to 1000 percent [4, 5].

Scheduled tasks are tasks that are executed at a specific point in time and that can repeat at scheduled intervals without explicit people intervention. Big servers (clusters, cloud and distributed systems etc.), personal computers and their operating systems used this concept for a long time. Even mobile systems use some form of these tasks. This wide usage is based on the fact that people can write and scheduled tasks without their explicit presence (send newsletter, periodic info, extract analytics data, run diagnostic tools etc.). Idea behind scheduled task is not new. This was one of tools that was available and originally developed on UNIX operating systems by Ken Thompson.

DSLs for scheduled tasks can increase productivity and reduce learning curve, especially if people wrote these tasks in some programming language, or if some DSL exists but they are not user-friendly. Tasks are pieces of program code that need to be executed at some specific time, together with its meta-data. Main motivation for the development of Kronos DSL was the ability to write tasks description in a language that is easy to read, write and learn.

This paper is organized as follows. Section II present design and implementation of Kronos Domain Specific Language. Section III present related work. Section IV summarize conclusions and briefly propose ideas for future work.

II. DESIGN AND IMPLEMENTATION

Kronos DSL is developed with textX [9]. textX is a meta-language (i.e. a language for language definition) for DSL specification in Python developed at Chair of Informatics, Faculty of Technical Sciences. textX is based on the Arpeggio PEG parser [10]. From a single grammar description textX dynamically builds a meta-model (in the form of Python classes) and a parser for your language. Parser will parse expressions on the defined language and automatically build a graph of Python objects (i.e. the model) corresponding to the meta-model. It has the following additional features: 1) automatic linking - textual references to other objects in the language will be resolved to proper python references automatically, 2) model/object post processing - callbacks (so called processors) can be registered for models and individual classes, which enables model/object post-processing (validation, additional changes etc.), 3) grammar modularization / imports - grammar can be split out into multiple files and then files/grammars can be imported where needed, and 4) meta-model/model visualization, both meta-model and parsed models can be visualized using the GraphViz software package [11].

Kronos is a DSL for scheduled tasks that is platform independent, and currently allow creating and running tasks in the background. It also has a possibility to call web services even for different versions, passes security keys for authentication, adds priority to every task, even ability to sync tasks that access shared data. In future Kronos can be extended with generator for Cron service or for various other scheduled tasks code-based solutions, and calling different operations that are not on web.

Solution is designed to be as close as it can to plain English and to eliminate unnecessarily programming languages constructs. With this is mind, users can easily write their own tasks, and leave it to Kronos to handle everything else.

A. Kronos meta-model and grammar

The grammar of the Kronos DSL, available at [12], is defined using textX. With the help of textX it is possible to create a meta-model of the Kronos language. Figure 2 describes the meta-model, shown as an UML-like class diagram generated by textX. Each class in the figure represents one textX rule, which is translated to a Python class at run-time. There is no code-generation involved. textX works as an interpreter. Association links in the diagram represent textX match rule references or link rule references.

Match rule references have containment semantics where the contained object (child) can exists only if its container (parent) object exists. Link rules don't have this restriction.

The core concept of Kronos DSL is Kronos rule which starts with *kronos:* keyword and contains one or more *Job* rules that contain all the information needed for proper task execution. Only few rules are mandatory, and the rest are optional. The user must specify short description (plain text, mainly for logging reasons), defined in the *Description* rule, *URL* for service to be executed (*URL* rule) and the time interval for task execution (*Scheduled* rule). Optionally, users can specify: 1) priority to task execution over the others specified tasks (value of 1 is assumed if not given), 2) security key in the form of authorization and/or authentication token (empty string is assumed if not given), 3) ability to sync tasks that access shared data, 4) and service version.

Figure 1 presents textX grammar of Job rule, and its usage in the task example, with Kronos header.

```
kronos:
  -description: "some scheduled task"
   url: /api/v1/foo/bar/
   1st mon of sep,oct,nov 17:00
   target: v1
   security: "AVBGFTTY"
   sync: true
   priority: 1
```

```
Job:
    desc = Description
    url = Url
    schedule = Schedule
    (target = Target)?
    (secure = Secure)?
    (sync = Sync)?
    (priority = Priority)?
;
```

Figure 1. Example of Kronos task definition with Job textX rule

There can be a lot of tasks, and the best practice is to separate tasks with one empty line, and every new task to mark with *"-"* symbol in description part. These are recommendations, not rules and they are optional for convenient reading. It is possible to put one line comments for every task defined by *Comments* rule. These comments contain plain text, and they are ignored by the interpreter. With defined types: *Ordinal*, *Month, Day, Unit, Time, TimeSpecial* we are able to define a complex set of intervals when to execute scheduled tasks. Every rule describes tasks that executes: 1) daily every n hours or minutes in from-to pattern (Figure 3), 2) in combination with *Synchronized* rule, users can easily define pattern that repeat during whole day. In both cases tasks is executed in a range of n hours or minutes, but minimum amount is one minute. For shorten writing, users can use *mins* instead of *minutes*.

On the other hand, *Selective* rule is designed for tasks that happened, not that often, or for some more complex patterns that needs to be used (Figure 4). Selective rule must be defined in the form: *<ordinal number list> <day list> <month list> <when to run>*.

```
description: "Some complex task"
url: /api/v2/foo/bar/
1st,second mon,wed of sep,oct every 2 hours from
17:00 to 22:00
```

Figure 4. Selective rule defines more complex pattern.

For example, if task need to be executed at the first and the second Monday and Wednesday of September and October at specific time, or even in from-to pattern in some range, every n hours or minutes, this rule can offer that functionality. This ordinal numbers can be represented via text: first, second, third etc., or in ordinal representations: 1*st, 2nd, 3rd etc*. Kronos is able to detect more Months, Days, and Ordinal numbers but they must be separated with *","* symbol. Also, users can use every keyword, if they what to specify that tasks run every day, month or ordinal respecting its place of usage. If the task runs whole day, Synchronized rule can be applied here too.

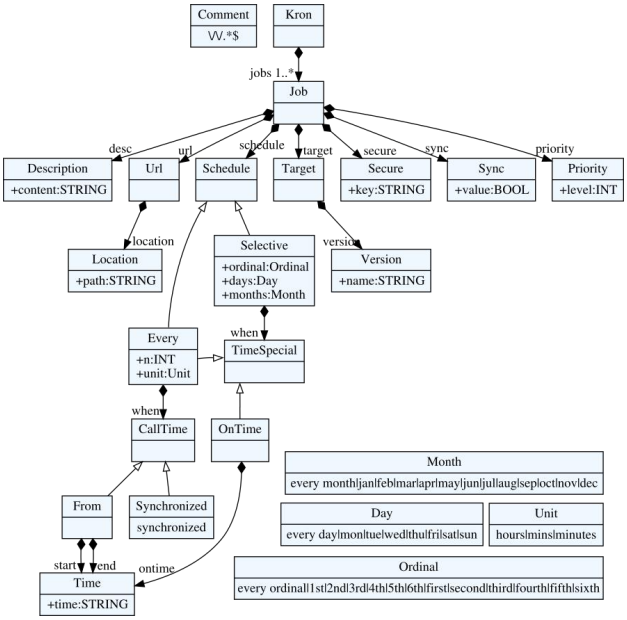

Figure 2. Kronos meta-model generated by textX from the grammar

```
-description: "Every task kind"
    url: /some/defined/url
    every 3 hours from 17:00 to 20:00
```

Figure 3. Every rule defined to run every 3 hours from 17-22h

Kronos grammar is separated in parts on the logical level using textX grammar modularization, and can be easily extended. The language itself is designed to be easy and intuitive for users, to provide plenty of opportunities, but also to be simple at the same time.

III. RELATED WORK

Scheduled task programs are usually web (service) or native, depending on the way how they run. Web tasks are not installed directly on the user side but run on the browser or on some cloud service. These tasks are platform independent. Usually, users, interact with them over some predefined interface (configuration file, APIs, service call etc.).

Native tasks are usually run on the user side, and they are integrated in operating system itself. They are not cross platform, although they have a significant performance boost. Usually, they do not require an internet connection in order to run.

Cron [6] is a time-based job scheduler in Unix-like computer operating systems, to run periodically at fixed times, dates, or intervals. It typically automates system maintenance or administration, though its general-purpose nature makes it useful for things like

downloading files from the Internet and downloading email at regular intervals.

Google App Engine Cron Service [7] allow users to configure regularly scheduled tasks that operate at defined times or regular intervals. These cron jobs are automatically triggered by App Engine Cron Service. For instance, you might use this to send out a report email on a daily basis, to update some cached data every 10 minutes, or to update some summary information once an hour. A cron job will invoke a URL, using an HTTP GET request, at a given time of day. An HTTP request invoked by cron is subject to the same limits as other HTTP requests, depending on the scaling type of the module. Free applications can have up to 20 scheduled tasks. Paid applications can have up to 100 scheduled tasks.

Csharp Schtick [8] a scheduled task runner built on Schyntax (a domain-specific language for defining event schedules in a terse, but readable, format)

APScheduler [13] is a Python library that lets you schedule your Python code to be executed later, either just once or periodically. You can add new jobs or remove old ones on the fly as you please. If you store your jobs in a database, they will also survive scheduler restarts and maintain their state. When the scheduler is restarted, it will then run all the jobs it should have run while it was offline.

IV. CONCLUSION

Since readily available, cross-platform DSLs for scheduled tasks that are easy to learn and read are not available to the best of our knowledge, we made a contribution to that area with Kronos. Kronos DSL uses declarative plain English-like syntax to describe job schedule specification.

Disadvantage of the current implementation is that every time user adds new task, Kronos needs to be stopped and started again, in order for the new task to be applied. Although, there is a possibility to add new tasks without restarting Kronos using API (Python code) it is still not very user-friendly.

Currently, Kronos is able to run both Every and Selective Tasks, but optimizations on sleeping and calculating when to run is done only for Every tasks. This has implication that Selective tasks will get checked every second until their run time comes.

Plans for further development of the Kronos DSL include: 1) create better optimizations for sleep time of Selective tasks, 2) ability to add new tasks without stopping currently active tasks, 3) extending grammar and implementing ability for tasks to call local code, instead just web services, 4) synchronize tasks that access shared data, and 5) integrating code generator for generating scripts for other similar tools that are more complex to use 6) design plugins for modern text editors like vim, emacs, sublime, or implementation of GUI tool, 7) implement to work in distributed envirement [14].

REFERENCES

[1] M. Simic, Kronos scheduled task dsl (2016), URL https://github.com/MilosSimic/Kronos, last accessed 5 April 2017

[2] M. Mernik, J. Heering, and A. Sloane, ACM Computing Surveys (CSUR) 37, 316–344 (2005).

[3] M. Fowler, Domain-Specific Languages, Addison-Wesley Professional, 2010.

[4] T. J.-P. Kelly S., Domain-Specific Modeling: Enabling Full Code Generation, Wiley-IEEE Computer Society Pr, 2008.

[5] Metacase, Nokia case study, Tech. rep. (2007).

[6] Cron (2016), URL http://pubs.opengroup.org/onlinepubs/9699919799/utilities/crontab.html, last accessed 5 April 2017.

[7] Google, Google app engine cron service (2016), URL https://cloud.google.com/appengine/docs/python/ config/cron, last accessed 8 April 2016.

[8] C. Schtick, C# schtick (2016), URL https://github.com/schyntax/cs-schtick, last accessed 5 April 2017.

[9] I. Dejanovic, Textx (2016), URL https://github.com/igordejanovic/textX, last accessed 1 March 2017.

[10] I. Dejanovic, G. Milosavljevi ´ c, and R. Vaderna, ´ Knowledge-Based Systems 95, 71 – 74 (2016), ISSN 0950-7051, URL http://www.sciencedirect.com/science/article/pii/S0950705115004761.

[11] Graphviz, Graphviz (2016), URL http://graphviz.org/, last accessed 7 April 2017.

[12] M. Simic, Kronos scheduled task dsl grammar (2016), URL https://github.com/MilosSimic/Kronos/tree/master/grammar, last accessed 2 April 2016.

[13] APScheduler, https://apscheduler.readthedocs.io/en/latest/https://apscheduler.readthedocs.io/en/latest/, Accessed 4 april 2017

[14] K Ousterhout, P Wendell, M Zaharia, I Stoica, University of California, Berkeley, https://cs.stanford.edu/~matei/papers/2013/sosp_sparrow.pdf, accssed april 4 2017