System Dynamics Modelling with Kotlin: From Hierarchical Models to Interactive Simulators

Siniša Sovilj
Juraj Dobrila University of Pula, Faculty of Informatics
HR-52100 Pula, CROATIA
http://hr.linkedin.com/in/ssovilj
sinisa.sovilj@unipu.hr

Darko Etinger
Juraj Dobrila University of Pula, Faculty of Informatics
HR-52100 Pula, CROATIA
darko.etinger@unipu.hr

Zlatko Sirotić
Istra Tech d.o.o.
HR-52100 Pula, CROATIA
zlatko.sirotic@istratech.hr

Krešimir Pripužić
University of Zagreb, Faculty of Electrical Engineering and Computing
HR-10000 Zagreb, CROATIA
kresimir.pripuzic@fer.hr

Keywords: *system dynamics, kotlin, modelling, complex system, hierarchical model, simulator*

Funding Source/Alternate Affiliations: This work has been supported in part by Croatian Science Foundation under the project UIP-2017-05-9066.

Extended Abstract: System dynamics (SD) modelling is traditionally done by graphically using mostly proprietary software or with couple of open and free software exceptions. Building large hierarchical models with reusable SD modules is not supported well and building fast and customizable simulators for interactive learning on multiple target platforms is not trivial.

There are couple of framework/toolkit solutions that fulfill that gap using general-purpose languages (Java, Python, JavaScript) but they have their own limitations. We asked whether Kotlin, as a new, modern, statically-typed, null-safe, object-oriented and functional language can do any better and overcome limitations of other programming languages and frameworks/toolkits. Therefore, we started to develop *Kotlin SD Toolkit* which does not exist so far at our best knowledge.

We found that Kotlin as a new, programming language and our Kotlin SD Toolkit as a new tool are both suitable for modelling larger, hierarchical SD models that support modules (see Table 1) and for easier development of interactive simulators for multiple target platforms: desktop, web or mobile (see Figures 1, 2 and 3).

We also measured speed properties without any a priori code optimizations and for now Kotlin SD Toolkit on our computer needs 7 seconds to numerically integrate simple testing SD model with 1E7 time steps, which is very fast in comparison with others and can be additionally improved.

We will continue to further expand capabilities of the Kotlin SD Toolkit as an open source project and our contribution to SD community.

Table 1. Kotlin code for Innovation/Product diffusion model (also known as Bass diffusion model).

Steps	Kotlin code

0) Setup	`// Static properties (optional)`
	`companion object {`
	`  const val TOTAL_POPULATION_VALUE = 10000  // [customer]`
	`  const val ADVERTISING_EFFECTIVENESS_VALUE = 0.011  // [1/year]`
	`  const val CONTACT_RATE_VALUE = 100  // [1/year]`
	`  const val ADOPTION_FRACTION_VALUE = 0.015  // []`
	
	`  const val INITIAL_TIME_VALUE = 0  // [year]`
	`  const val FINAL_TIME_VALUE = 10  // [year]`
	`  const val TIME_STEP_VALUE = 0.25  // [year]`
	`}`

1) Model	`init {`
	`  val model = Model()`
	
	`  // Override default model properties`
	`  model.initialTime = INITIAL_TIME_VALUE`
	`  model.finalTime = FINAL_TIME_VALUE`
	`  model.timeStep = TIME_STEP_VALUE`
	`  model.integration = EulerIntegration()`
	`  model.name = "Innovation/Product Diffusion Model" // optional`

2) Entities	

- Constants	`  val TOTAL_POPULATION = model.constant("TOTAL_POPULATION")`
	`  val ADVERTISING_EFFECTIVENESS =`
	`  model.constant("ADVERTISING_EFFECTIVENESS")`
	`  val CONTACT_RATE = model.constant("CONTACT_RATE")`
	`  val ADOPTION_FRACTION = model.constant("ADOPTION_FRACTION")`

- Converters	`  val adoptionFromAdvertising =`
	`  model.converter("adoptionFromAdvertising")`
	`  val adoptionFromWordOfMouth =`
	`  model.converter("adoptionFromWordOfMouth")`

- Stocks	`  val Potential_Adopters = model.stock("Potential_Adopters")` `  val Adopters = model.stock("Adopters")`
- Flows	`  val adoptionRate = model.flow("adoptionRate")`
- Modules	
3) Initial values	
- Stocks	`  Potential_Adopters.initialValue = { TOTAL_POPULATION }` `  Adopters.initialValue = { 0.0 }`
4) Equations	

- Constants	`  TOTAL_POPULATION.equation = { TOTAL_POPULATION_VALUE }`
	`  ADVERTISING_EFFECTIVENESS.equation =`
	`  { ADVERTISING_EFFECTIVENESS_VALUE }`
	`  CONTACT_RATE.equation = { CONTACT_RATE_VALUE }`
	`  ADOPTION_FRACTION.equation = { ADOPTION_FRACTION_VALUE }`

- Converters	`  adoptionFromAdvertising.equation =` `  { Potential_Adopters * ADVERTISING_EFFECTIVENESS }`

	adoptionFromWordOfMouth.equation = { CONTACT_RATE * ADOPTION_FRACTION * Potential_Adopters * Adopters / TOTAL_POPULATION }
- Stocks	Potential_Adopters.equation = { - adoptionRate } Adopters.equation = { adoptionRate }
- Flows	adoptionRate.equation = { adoptionFromAdvertising + adoptionFromWordOfMouth }
- Modules	
5) Simulation	val simulation = Simulation(model)
6) Outputs	simulation.outputs {
- Text	CsvExporter("output.csv", ";"))
- Image	PngExporter("chart.png"))
- Desktop	WinSimulator()
- Web	WebSimulator()
- Mobile	MobSimulator()
	}
7) Run	simulation.run()
	}

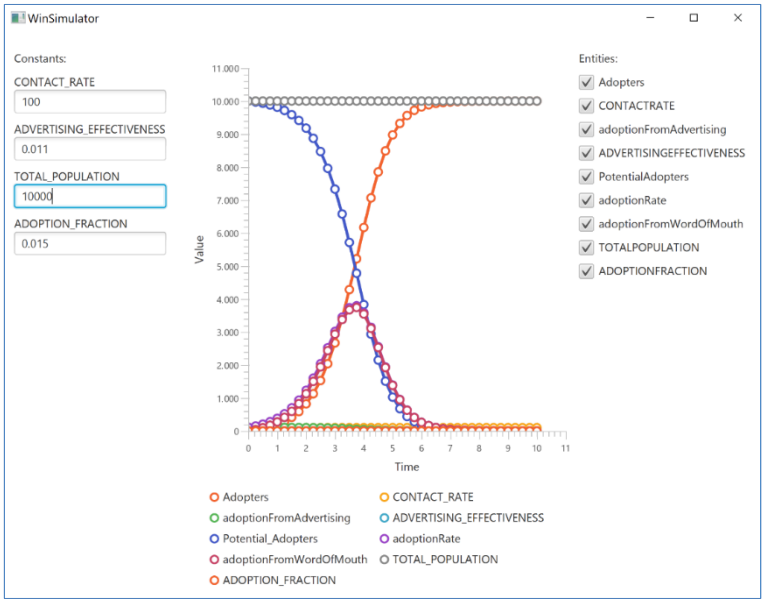

Figure 1. Desktop Simulator – an interactive simulation environment of the Bass diffusion model. The interactive simulation environment is a desktop application window and is automatically generated based on entities' type (we can change constants interactively, and we can enable or disable plots of any model's entities).

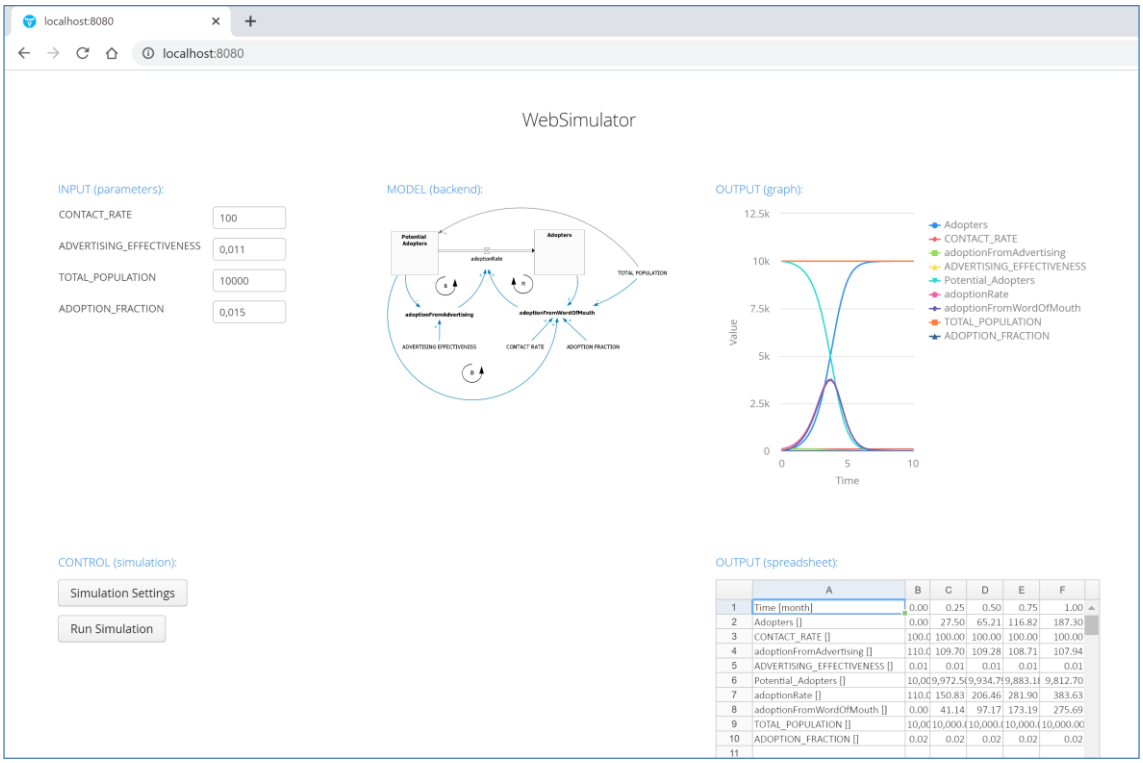

Figure 2. Web Simulator – an interactive simulation environment of the Bass diffusion model. The interactive simulation environment is a web application rendered in any Internet browser and is automatically generated based on entities' type. We can change model constants, simulation settings and display the model diagram as an option.

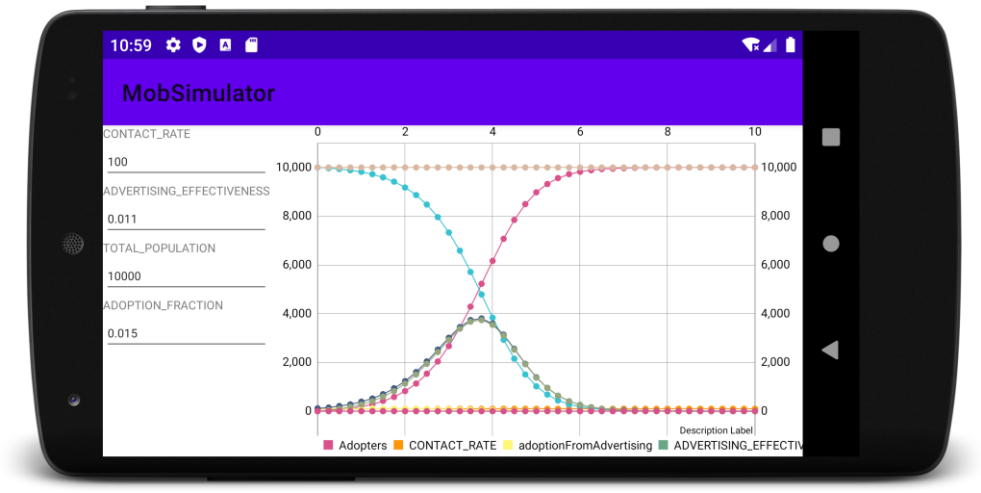

Figure 3. Mobile Simulator – automatically generated interactive simulation environment for Bass diffusion model. The interactive simulation environment is an Android mobile application. We can change model constants and re-run the simulation on a mobile phone.

References

[1] Simantics Team, February 2018. [Online]. Available: https://www.simantics.org/end_user_wiki/index.php/Simantics_System_Dynamics#Installation_Instructions.

[2] F. Lardinois, "Kotlin is now Google's preferred language for Android app development," May 2019. [Online].

[3] S. Drost and M. Stein, "System Dynamics Java-Framework," February 2017. [Online]. Available: https://github.com/matthiasstein/SystemDynamics-Framework.

[4] D. Schroeck, "Business Prototyping Toolkit for Python (BPTK-Py)," January 2020. [Online]. Available: https://pypi.org/project/BPTK-Py/.

[5] B. Powers, "sd.js. open source SD," Spetember 2019. [Online]. Available: https://sdlabs.io/.

[6] B. Powers, "Simulation Speed: Fast," September 2019. [Online]. Available: https://sdlabs.io/speed/.