Simulation in Production
and Logistics 2015
Markus Rabe & Uwe Clausen (eds.)
Fraunhofer IRB Verlag, Stuttgart 2015

A Natural-language-based Simulation Modelling Approach

Mit natürlicher Sprache zum Simulationsmodell

Thomas Mayer, Falk Stefan Pappert, Oliver Rose, Universität der Bundeswehr München, Neubiberg (Germany), thomas.mayer@unibw.de, falk.pappert@unibw.de, oliver.rose@unibw.de

Abstract: Formal methods to describe complex production and logistic systems for the purpose of simulation are increasing in importance. Looking at modern software development methodologies, i.e., Behaviour Driven Development (BDD), we see promising approaches to requirements analysis. BDDs principles are based on the idea of automated execution of requirement specifications which are formulated in semi-formal, executable natural language constructs, thereby improving the communication between stakeholders and development team. In this paper, we introduce natural language constructs for modelling systems from the domain of production and logistics. Their usage is showcased with the help of an example production system and their usability and suitability are evaluated with respect to the modelling process. Furthermore, challenges in the use of natural language as a new formalism to support simulation modelling are discussed.

1 Introduction

One of the main tasks in almost every simulation project is the development of a valid simulation model, To this end, conceptual modelling is one of the major aspects (Robinson et al. 2011). The common modelling and validation approaches emphasise the importance of communication between experts and stakeholders (Sargent 1996; Rabe et al. 2008). Pace (2000) also highlights the problem of understanding of what solution to implement. A starting point to support the communication between experts and stakeholders is a suitable representation of the problem. Robinson (2006; 2013) identifies the need of research in developing methods for designing conceptual models. Huang et al. (2007) approached this topic by using SysML to visualize processes and structures of a real world system. Another graphical approach was introduced by Ahmed at al. (2014) into the domain of Discrete Event Simulation using the Structured Analysis and Design Technique (SADT). SADT is an established concept in software engineering, specifically during the requirements analysis. In general, there is a similar communication challenge in the process of developing software. There, the software developer or archi-

tect has to understand the requirements for the product and the stakeholder need to be able to validate the delivered software based upon the agreed-on requirements. Beside graphical approaches it is common best practice to formalize requirements in natural language to increase understanding (Soeken et al. 2012). Furthermore, formalized executable requirements are used to automate the validation process. A well-known natural-language-based requirement formalization approach in software engineering is implemented within the software testing tool Cucumber. Cucumber automates acceptance testing based on requirements formalized in the language Gherkin that aims to use natural language constructs (Wynne and Hellesøy 2012). A broader approach was introduced as "Lean Modelling" (Seifert et al. 2013).

We investigate whether it is feasible to adopt this non-graphical requirements engineering approach based on natural languages to describe systems for modelling and simulation. Furthermore, it is evaluated whether a non-graphical approach is a suitable representation of a system. Therefore, language constructs are developed, which are based on the ideas behind Gherkin. A transformation into a simulation model is implemented that can be executed in our own simulation environment.

From our point of view, there are four mandatory components in every simulation environment (fig. 1). The editor component is responsible for user interaction, nowadays usually a graphical editor for model development which is providing support functionality to improve usability. With the help of the editor the conceptual model is implemented into a custom model, which represents the real world model within a tool-specific formalization and therefore the metamodel of the editor. The custom model is transformed into a simulator model that is based on the simulation tool's metamodel. This metamodel is the inherent model specification description of a simulator.

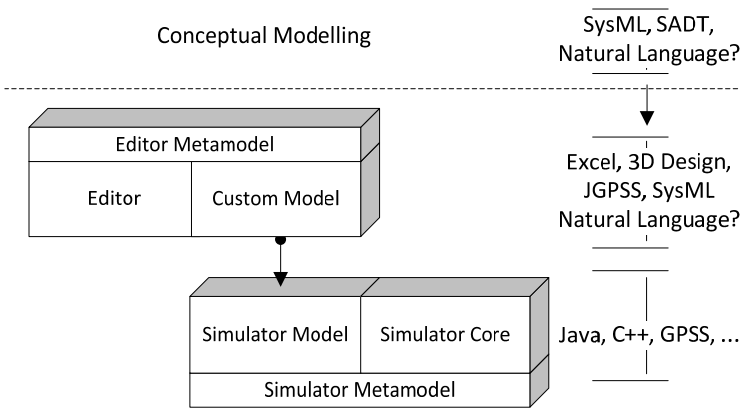

Figure 1: *Component model*

The presented natural language modelling approach focusses first on the conceptual modelling phase and second on the components editor and custom model. The editor component is a simple text editor for natural language constructs to create the custom model. After transformation into the simulator model, it is executed by the simulation core. Metamodel and simulation core are implemented in Java.

In the following, we will first give an insight into Gherkin and the ideas behind it. A small production system is shown which is used as a first test case. In Section 4 our language constructs are introduced. The transformation process of our language constructs into our metamodel is addressed in Section 5. Conclusions are drawn in Section 6.

2 The Gherkin Language

In Behaviour Driven Development (BDD), a software development approach building upon the foundation of Test Driven Development (TDD), there is a focus on involving every member of the development team with the goal of the software (Wynne and Hellesøy 2012). This focus is to ensure that only valid features are implemented and to have everyone from the stakeholder to the developer to talk on a level playing field. This should be done in the same language, thereby avoiding common problems of misunderstanding and miscommunication. Although certain experts will still focus the main part of their work in their area of expertise everyone in the team is supposed to be able to read and understand all documents of the project. To ensure that this approach is possible, the language in which specification and documentation of the system are written in needs to be ubiquitous. A further step to clarify specifications is to write them in an executable way. Thereby, specifications can be used to directly test and evaluate the state of the software system development without the need for further interpretation and translation into a testable format by a test professional. A well-known technology in the area of software development to implement this approach is Cucumber with the underlying language Gherkin. It is used to communicate, discuss, document and later automate specifications of the software in development.

Features are described with the help of use cases in the language Gherkin that is very close to natural language. Algorithm 1 shows a small Gherkin example of a calculator.

Algorithm 1: *Gherkin example*

```
Feature addition of two numbers.
Scenario basic addition.
     Given the calculator is open
     When I enter the number 3
     And I press the plus button
     And I enter the number 4
     Then I should see the number 7 as result.
```

There are only very few meta symbols and key words used. They are the first words in every line. The majority of the specification is actually close to real English sentences. This clear and simple way of describing features is, therefore, easy to understand to all project members. In contrast to other specification approaches, the major attention in Gherkin is set on having a simple and general language with only a few key words and an intelligent mapping of semantics. The interested reader can find a discussion on the benefits of using BDD and Gherkin as well as thoughts on the completeness of this language construct in (Martin 2008). As mentioned before,

the focus is not only on having a simple natural-language-based specification. Rather there is significant emphasis on being able to execute these specifications automatically as tests when the features are implemented by the software developers in the team. This automation is done by mapping sentences of the generated specification to methods and functions of the source code.

The language Gherkin is a very useful tool to tackle descriptions of use cases. Although, it is not possible to describe a manufacturing system only with use cases and expect an algorithm to generate a valid simulation model of the described system. We therefore see the necessity in a language to support more than simply describing use cases. There is in fact the need to describe structure and general behaviour of a system to be able to generate a simulation model.

3 Forklift Example

It is a common practise to evaluate the capabilities of a modelling approach starting with a small model from the intended domain. We have chosen a well-known model from the domain of production and logistics from Law and Kelton (2000). It is illustrated in Figure 2. The system modelled is a small production system utilizing forklift trucks for transport. It consists of five processing areas (work station) and one area for receiving and shipping. Within each production area there are a number of equivalent machines, which share a single queue. Three different types of jobs move through the system, differing mainly in processing times and order of their tasks.

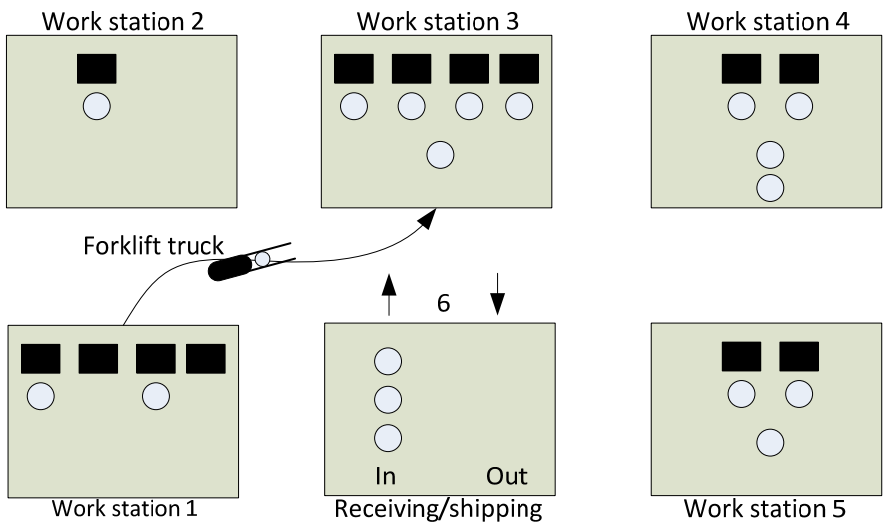

Figure 2: *Small production system with forklift trucks (Law and Kelton 2000, p. 685)*

4 Our New Natural-language-based Modelling Approach

In the following chapter, our basic language constructs are introduced with the help of the small example production system presented in Section 3. We start with describing the structure and then move on to behaviour.

4.1 Representation of Structure

Looking at the simple sentence "There is a tree in the forest, it has green leafs.", real world objects are intuitively addressed with the phrase "There is a" followed by the name or type of the object. Hierarchical relations between objects can be described with "… in …" and properties are formulated with "… has a …". Following this simple sentence we derive our language construct for modelling structure. Algorithm 2 shows an example of the structural constructs. It demonstrates the three structural elements: `Model`, `Workstation` and `Queue`. The `Workstation` with the name `WS1` is a sub-component of `Model M1`. It has two properties with the name `capacity` and `content`. These properties have initial values of 3 and 0 respectively. In the "has a …" statement the object is not directly addressed again, it is derived from context. Every "There is a …" phrase creates a context which can be used by subsequent phrases. Type and object names are used as identifiers, which an object is referred to globally. All structural elements can be modelled with these two phrases.

Algorithm 2: *Structure elements with properties*

```
There is a Model M1.
There is a Workstation WS1 in M1.
     has a capacity of 3.
     has a content of 0.
There is a Queue WS1_Q1 in WS1.
     has a capacity of 100.
     has a content of 0.
```

Algorithm 3 illustrates the representation of distances between the workstations from the small production system presented in chapter 3. Each distance is modelled as a property in the context of the structural element `TransportMatrix`. In common practise distances are not described in this way but in our feasibility study we use this approach as shortcut to avoid increased implementation effort.

Algorithm 3: *Structure element TransportMatrix*

```
There is a TransportMatrix TM1 in M1.
          has a WS1_WS2 of 150
          has a WS1_WS3 of 213
```

The flow through the production system is represented by additional phrases, which are describing job types. They are modelled from the viewpoint of the flow item. In

our sample production system a flow item can use machines and forklifts for a specified time and change their properties, e.g. a forklifts location. These resource allocations and state changes are modelled with the phrase "uses … for … from … to …". The variable elements of the phrase represent the allocated object, duration, and locations.

Algorithm 4: *Structure element JobType*

```
There is a JobType JT1 in M1.
      is created in RS1_Source every 5
      uses RS1_Q1
      uses FL1 for TM1.(FL1.location)_RS1/FL1.speed
            from FL1.location to RS1
      uses FL1 for TM1.RS1_WS3/FL1.speed from RS1 to WS3
      uses WS3_Q1
      uses WS3 for 0.25
```

Algorithm 4 illustrates the modelling of the structural element `JobType` with the name `JT1` as sub-component of `Model M1`. A flow item is created every 5 time units in the structural element that has the name `RS1_Source`. Every "uses …" phrase refers to one structural element to be allocated. Properties of other structural elements can be referenced by the element name and its properties name concatenated with a dot. This way of referring to properties seems more common for programming languages than for natural language, but is chosen for this demonstration as it is easier to handle for our current tool chain. With an improved tool chain it is possible to reference properties in a more natural way. Gelhausen (2010) is offering ideas on interpreting natural language which we are going to consider in future research.

4.2 Representation of Behaviour

While moving through the system a flow item is using different structural elements, which can change properties. Furthermore, certain conditions to use objects must be considered. In the sample production system a single machine can only process one flow item at a time.

Figure 3 illustrates the concept of our behaviour model. Only if the `enter condition step 1` is met, the `usage state step 1` is entered. As long as the succeeding structural enter condition of the element is not met, the flow item is kept in a loop between the states `after usage step 1` and `undo after usage step 1`. This loop is a simple way to get a full picture of the system state after leaving the current step. We therefore roll out all after usage changes to the system. In case the conditions to proceed are not met, the changes are simply rolled back to keep a consistent system state. This check loop is triggered again when relevant system changes have happened. The flow item can only leave the loop, if the enter condition for the following element is met, only by passing the `after usage state`.

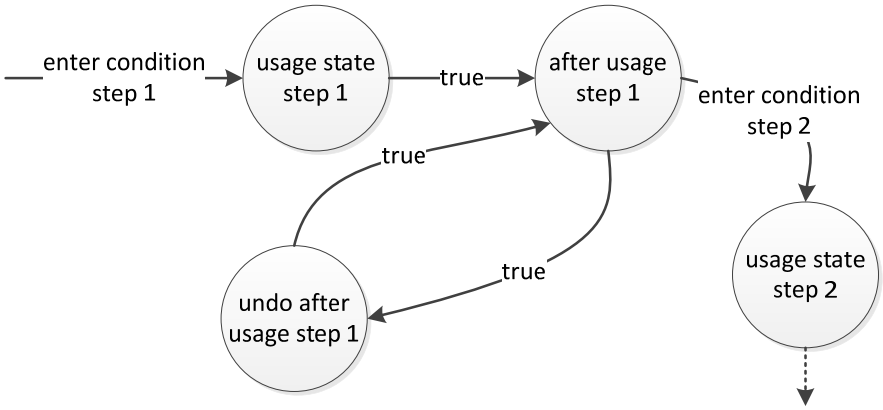

Figure 3: *Step processing state chart*

Algorithm 5 showcases the modelling of a workstation's behaviour. It is represented as `Businessrule` with the name `BR_W`. The same approach is used to access properties as in modelling structural elements. Additionally, mathematical operations and expressions to model conditions and state changes are supported. The delay of a flow item is modelled with the phrase "During use delay process for param1" and is dependent on the flow item. Values, given in the job type definition, can be accessed with the keywords "param0", "param1", "param2", and "param3". As an example, the JobType description in Algorithm 4 is considered again. The last step of the JobType is: "**uses** WS3 **for** 0.25". When a step is executed, businessrules can access its parameters. In this case "param0" is "WS3" the resource to be used and param1 refers to the duration of 0.25.

Algorithm 5: *Definition of behaviour*

```
There is a Businessrule BR_W for Workstation
     Before use check if content<capacity
     During use set content++
     During use delay process for param1
     After use set content--
     Undo after use set content++
```

5 Model Transformation

This section gives an overview on the technologies used to transform the model into executable code or into the model description of an arbitrary simulator. As mentioned before, there is a significant advantage in an automated transformation of a specification into a target model or program. In the case of Gherkin this transformation can be done with a number of tools. One of the most prominent ones is Cucumber. Since our natural language constructs differ from Gherkin not only in syntax but also in grammar it is not possible to do this mapping with the help of Cucumber. In fact a tool with a much broader scope is necessary. One such tool is

NatSpec from DevBoost GmbH. NatSpec is an Eclipse plugin, which is supporting the mapping of natural language phrases to Java methods (Seifert et al. 2013). The general idea behind these tools is to map clauses and sentences of a specification to methods of source code. During this mapping values of parameters are identified and handed to the target method. Then the method is executed.

In the following, this mapping is demonstrated for the "There is ..." clauses. Given our phrases NatSpec generates method headers to specify the mapping. The java method header for the "There is …" clause is shown in Algorithm 6.

Algorithm 6: *Generated method header for "There is ..." clause*

```
@TextSyntax("There is a #1 #2 in #3.")
public SimulationEntity thereIsA_In_
      (String clazz, String name, String contextId,
            SimulationEntity context) { … }
```

The annotation, `@TextSyntax("There is a #1 #2 in #3.")`, is the basis of the mapping, as it contains the fixed parts of a clause and the position of all parameters of the method. Sentences like `There is a Workstation WS1 in M1.` are now recognizable and are interpreted as a method call with their specific parameters. The values written in the parameter positions are mapped to the corresponding ones of the method. The actual method (not shown here) contains functionality to create the referred object, in our example a workstation object, within the model and relates it to `M1`, as defined in the third parameter. This third parameter and the return value of the method represent a very common feature in natural languages, the semantic context. Usually a sentence in natural language does not stand for its own but has some links and references to previous and following sentences to avoid repetition and improve the flow of a text. A method can take into account the previous context by utilizing the context parameter and can hand its own context to subsequent sentences. An example of sentences using context can be seen in Algorithms 2 the "There is …" clause creates the context of a new workstation. The following "has…" statements build on this context and define parameters without additionally mentioning the workstation object.

Although this might seem complicated, this is a quite simple process. The implementation of nine methods has been sufficient to transform all possible models that can be represented in our natural language constructs.

Another important part of the model is the actual behaviour within the business rules. Besides clauses, which are interpreted based on the previously discussed mapping approach; there are conditions and state changes to be evaluated during the simulation run. These statements are transformed into strings of Java code and executed during runtime using a framework called Javassist. Javassist is a framework that allows manipulating java bytecode during runtime. It can be used to generate and use new Java classes during the execution of a Java program. We consider Javassist to be a very interesting and powerful technology but, with respect to the scope of this paper no further technical detail is given here (see Chiba and Nishizawa 2003 for more information on Javassist).

In summary, every sentence is mapped to a parameterized method call which creates parts of the model during its initialization. Behaviour is generated during model execution using Javassist.

6 Conclusion and Outlook

Based on the ideas of requirements engineering and BDD, we introduce a modelling approach based on natural language. The main idea is to utilize sentences in natural language to improve the communication between stakeholders and simulation experts during the early stages of simulation projects by formalizing specifications in a way that comes natural to everyone involved. We pointed out the benefits of writing executable specifications, which in an ideal case should be automatically translated into a simulation model of an arbitrary simulation system. The main benefit is the reduced need for interpretation of requirements by project members who are not domain experts. In addition, we see benefits for model validation by reducing miscommunication and helping to improve understanding of the real world system and its model. As this is only a first test case there are several open issues to be addressed in future research.

6.1 Extend the Language Construct with more Production-related Information

Currently, our language constructs only capture some major features of production and logistics systems. Although, they are able to show case the possibilities of the idea behind the modelling approach, we do feel it is necessary to include more production details to be able to model real systems.

6.2 Moving the Language Closer to a Natural Language

Looking at the way we are currently describing behaviour, there surely is some work to do before it is close to proper natural language. As mentioned before those statements which are still similar to source code can be easily replaced by natural language within the constructs but their interpretation and automated execution demand a tool chain with more capabilities than our current one. We therefore see the need to improve our toolchain and the tools within it to reach a language quality which is no longer cryptic to non-IT professionals. Due to the trend of language-based assistants in modern electronic gadgets and smartphones there is a broad interest in technologies that process and interpret natural language. We aim to utilize this new research to further improve the quality with which models can be developed and communicated.

6.3 Development of Editor Support

There is a need for a good development environment for modelling. Looking at modern software development and its integrated development environments we see enormous benefits in adapting common practises like auto-completion and automated error reporting to improve the usability of an environment and to help users develop their models.

References

Ahmed, F.; Robinson, S.; Tako, A.A.: Using the structured analysis and design technique (SADT) in simulation conceptual modeling. In: Tolk, A.; Diallo, S.Y.; Ryzhov, I.O.; Yilmaz, L.; Buckley, S.; Miller, J.A. (Hrsg.): Proceedings of the 2014 Winter Simulation Conference (WSC), Savannah (USA), 7.-10. December 2014, S. 1038-1049.

Chiba, S.; Nishizawa M.: An easy-to-use toolkit for efficient Java bytecode translators. In: Czarnecki, K; Pfenning F.; Smaragdakis, Y. (Hrsg.): Proceedings of the 2nd international conference on Generative programming and component engineering, Erfurt (Germany), 22.-25. September 2003, S. 364-376.

Gelhausen, T.: Modellextraktion aus natürlichen Sprachen. Karlsruhe: KIT Scientific Publishing 2010.

Huang, E.; Ramamurthy, R.; McGinnis, L.F.: System and simulation modeling using SysML. In: Henderson, S.G.; Billder, B.; Hsieh, M.-H.; Shortle, J.; Tew, J.D.; Barton, R.R. (Hrsg): Proceedings of the 2007 Winter Simulation Conference (WSC), Washington, D.C. (USA), 7.-10. December 2007, S. 796-803.

Law, A.M.; Kelton, W.D.: Simulation Modeling & Analysis. New York: McGraw-Hill, Inc. 2000.

Martin, R.C., 2008: The Truth about BDD. Accessed March 31. 2015. https://sites.google.com/site/unclebobconsultingllc/the-truth-about-bdd.

Pace, D.K.: Ideas about simulation conceptual model development. Johns Hopkins APL Technical Digest 21 (2000) 3, S. 327-336.

Rabe, M.; Spieckermann, S.; Wenzel, S.: Verifikation und Validierung für die Simulation in Produktion und Logistik. Berlin, Heidelberg: Springer 2008.

Robinson, S.: Conceptual modeling for simulation: Issues and research requirements. In: Perrone, L.F.; Wieland, F.P.; Liu, J.; Lawson, B.G.; Nicol, D.M. Fujimoto, R.M. (Hrsg.): Proceedings of the 2006 Winter Simulation Conference (WSC), Monterey (USA), 3-6 December 2006, S. 792-800.

Robinson, S.: Conceptual modeling for simulation. In: Pasupathy, R.; Kim, S.-H.; Tolk, A.; Hill R.; Kuhl M.E. (Hrsg.): Proceedings of the 2013 Winter Simulation Conference (WSC), Washington, D.C. (USA), 8-11 December 2013, S.377-388.

Robinson, S.; Brooks, R.; Kotiadis, K.; Van Der Zee, D.-J.: Conceptual Modeling for Discrete-Event Simulation. Boca Raton (FL): CRC Press 2011, S. 3-30.

Sargent, R.G.: Verifying and validating simulation models. In: Charnes, J.M.; Morrice, D.M.; Brunner, D.T.; Swain, J.J. (Hrsg.): Proceedings of the 1996 Winter Simulation Conference (WSC), Coronado (USA), 8.-11. December 1996, S. 55-64.

Seifert M.; Wende C.; Heidenreich F.; Nestler T.: Lean Modelling: Mit natürlicher Sprache zum Modell. Java Magazin 12.13 (2013).

Soeken, M.; Wille R.; Drechsler R.: Assisted behavior driven development using natural language processing. In: Furia, C.A.; Nanz, S. (Hrsg.): Objects, Models, Components, Patterns. Berlin: Springer 2012, S. 269-287.

Wynne, M.; Hellesøy, A.: The Cucumber Book: Behavior-driven development for testers and developers. USA: The Pragmatic Programmers, LLC 2012.