Building a Software Architecture out of User Stories and BDD Scenarios: Research Agenda

Samedi Heng[1], Monique Snoeck[2] and Konstantinos Tsilionis[2]

[1]*HEC Liège, Université de Liège, Liège, Belgium*
[2]*KU Leuven, Leuven, Belgium*

Abstract
User stories (US) are classically used as requirements engineering artifacts in agile methods like Scrum, these are sometimes associated with Behavior Driven Design (BDD) scenarios. Previous research allowed to unify US and BDD scenarios templates through the definition of a set of concepts from different nature traditionally used in instances of both of these concepts. When associated to these concepts, information is given on the nature of the US and BDD instances. So called high-level development languages provide strong abstractions on the basis of which software can be developed. These abstractions mimic human behavior at software runtime making the development process easier and understandable by human beings. Research has shown that functions of different nature defined in US and BDD instances do represent an interesting input to define a software architecture within Agent- and Object-Oriented (AO and OO) languages. While the mapping to AO is quite intuitive, the mapping to OO concepts is less one-on-one and requires a more in-depth analysis of the sentences that make part of the US and BDD definition. This can be done manually, but support by means of intermediate transformations or NLP is possible as well. This article summarizes the state of the art in the field and points to future work.

Keywords
User Stories, Behavior Driven Development, Acceptance Test, Software Architecture.

1. Introduction and Background

User Stories (US) are used to express requirements in structured natural language when developing software with the agile methods like Scrum. They document what functions or features a user wants the system to satisfy and are built around three complementary dimensions, i.e. WHO requires the functionality, WHAT the functionality is and WHY it is required/desired. Their main advantage is that they are easy to write and understand but lack the necessary details for designers and developers to effectively build the supporting software. Behavior Driven Design (BDD) scenarios can be associated with US to further document the concrete way a US should be realized by the software system. With the scenario, the development team gets the details on how the system should behave at runtime before, during at after the functionality is executed. In fact, the BDD scenario thus allows to validate the requirement. BDD scenarios are structured following three complementary dimensions, i.e. GIVEN a specific context, WHEN the functionality is executed, THEN the system is in a resulting state.

Agil-ISE 2022: Intl. Workshop on Agile Methods for Information Systems Engineering, June 06, 2022, Leuven, Belgium
samedi.heng@uliege.be (S. Heng); monique.snoeck@kuleuven.be (M. Snoeck);
konstantinos.tsilionis@kuleuven.be (K. Tsilionis)
0000-0002-6037-0914 (S. Heng); 0000-0002-3824-3214 (M. Snoeck); 0000-0001-9702-6941 (K. Tsilionis)

Both US and BDD scenarios contain relevant data and domain vocabulary useful for designing the software system. Indeed, Agent- and Object-Oriented (AO and OO) software mimic real life organizational behavior at application runtime to ease the implementation. Indeed, the intrinsic attributes and behavioral characteristics of AO and OO can partly be found in the expression of US/BDD. Also, meta-data furnished at modeling time on the type of element depicted in the US and the BDD scenarios furnish relevant information to map the requirement-level element with a design-level one.

Wautelet et al. [1] have investigated the US templates that are the most used in practice. Out of them they selected the keywords that were the most relevant and associated them to a particular semantic. Ultimately, a conceptual model made out of the most used concepts found in US templates has been built supporting the creation and furnishing useful meta-data on the nature of the depicted element when followed at modeling-time. This model is represented in Fig. 1; the definitions of each concept can be found in [1]. A similar investigation has been performed in Tsilionis et al. [2] out of the BDD templates and lead to the unified conceptual model presented in Fig. 2.

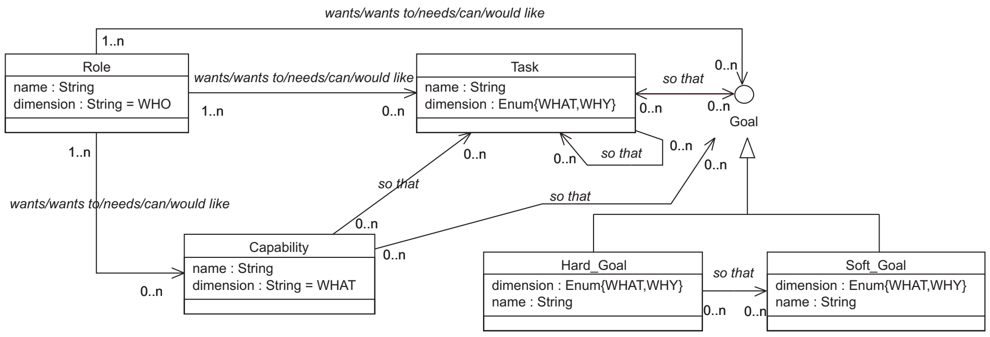

Figure 1: Unified Model for User Stories (from [1]).

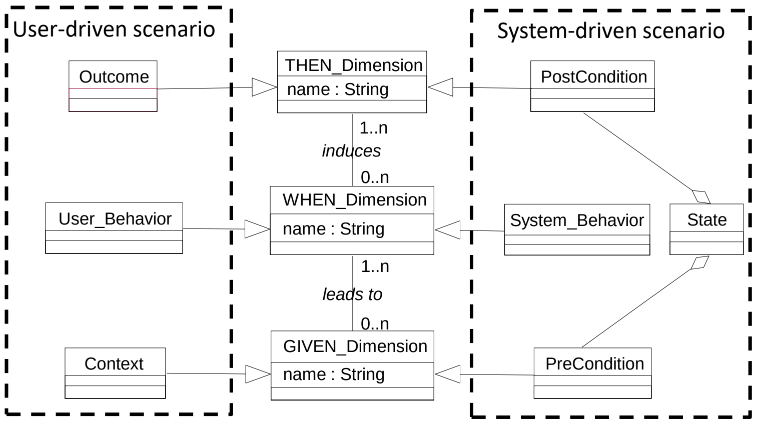

Figure 2: Unified Model for BDD Test Scenarios.

2. Building a Software Architecture From User Stories and BDD Scenarios

This section overviews how to transform the concepts presented in both US and BDD scenarios to an AO architecture (Sect. 2.1) as well as OO software (Sect. 2.2). For the agent architecture, we provided the tranformation for both Belief-Desire-Intention (BDI [3]) agent (using JaCaMo [4]) and none-BDI agent (using JADE [5]). We have chosen these high-level programming paradigms because the OO paradigm is the most used paradigm by many programming language such as Java, Python, etc., while, the AO paradigm is a new paradigm and it provides higher level abstraction compared to the OO paradigm. It allows to design software be more autonomous and social [6]. We focus on transforming the BDD System-Behavior scenarios to both AO and OO architecture. Sect.2.3 illustrates the transformation examples.

2.1. Agent-Oriented Architecture Transformation

2.1.1. Transforming to a None-BDI Agent Architecture (JADE framework)

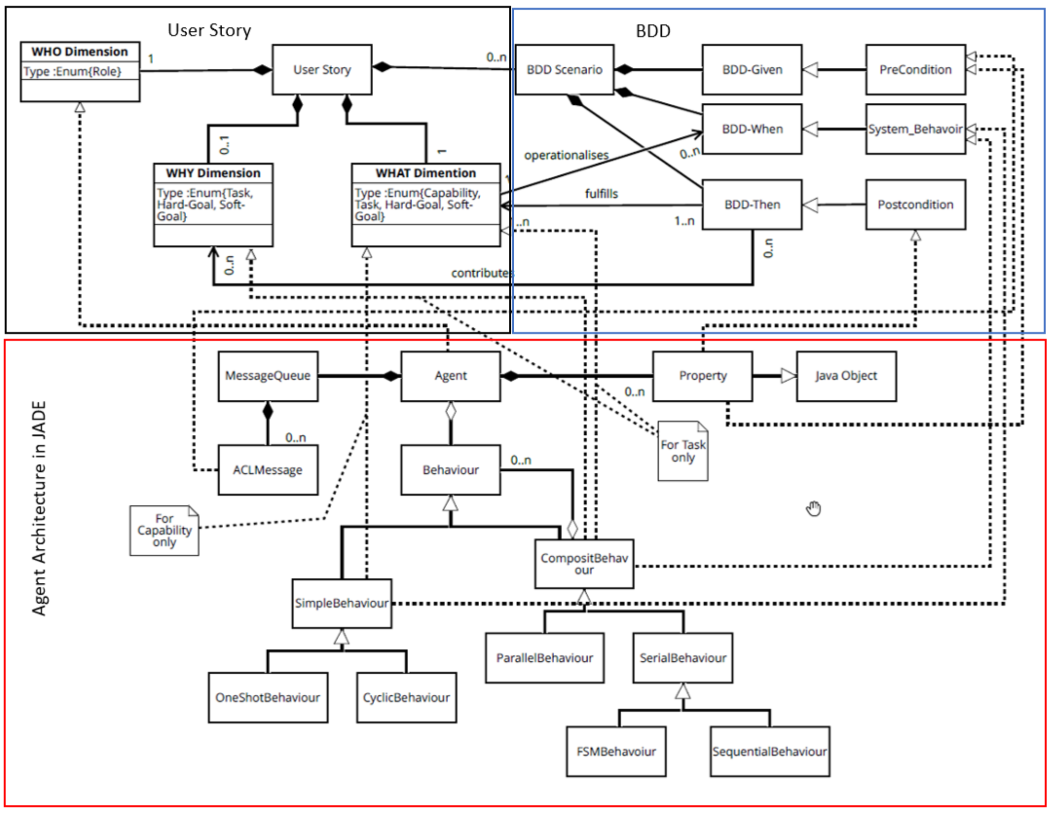

Figure 3: The mapping from user story and BDD to None-BDI Agent (JADE framework).

A JADE agent is composed of *Behaviour* and *MessageQueue* properties. The *MessageQueue* is used to store the communication message from others agents. The *Behaviour* is a task that an agent can carry out so it implements its possible actions. There are two types of Behaviour: *SimpleBehaviour* and *CompositeBehaviour*. The *SimpleBehaviour* is atomic, while the *CompositeBehaviour* can be composed of other behaviours (see Fig. 3).

With respect to US concepts, previous work [7] showed that the concept of *Role* in the WHO-dimension of a US can be mapped to an *Agent* in JADE. The concept of *Task* presented in both WHAT- and WHY-dimensions of the US are mapped into *CompositeBehaviours*. Finally, the concepts of *Capability* presented in the WHAT-dimension of US can be mapped to *SimpleBehaviours*.

With respect to the BDD concepts, the GIVEN- and THEN-dimensions do refer to states. They indeed describe state of a system or an agent rather then an action. Therefore, these concepts can be mapped to the *Property* characteristic of an agent which is a JAVA object. Since it is a state, it could represent the received message from another agent as well; hence, they could be transformed to an *ACLMessage*. In turn, the WHEN-dimension represents an action potentially performed by an agent. This concept can then be transformed to a *Behaviour* in JADE. The latter BDD dimension operationalizes the WHAT-dimension of a US, therefore, we argue that it should be implemented within the Behaviour of the US it operationalizes. Fig. 3 shows the mapping of US and BDD scenarios concepts to the architectural ones of JADE.

2.1.2. Transforming to a BDI Agent Architecture (JaCaMo framework)

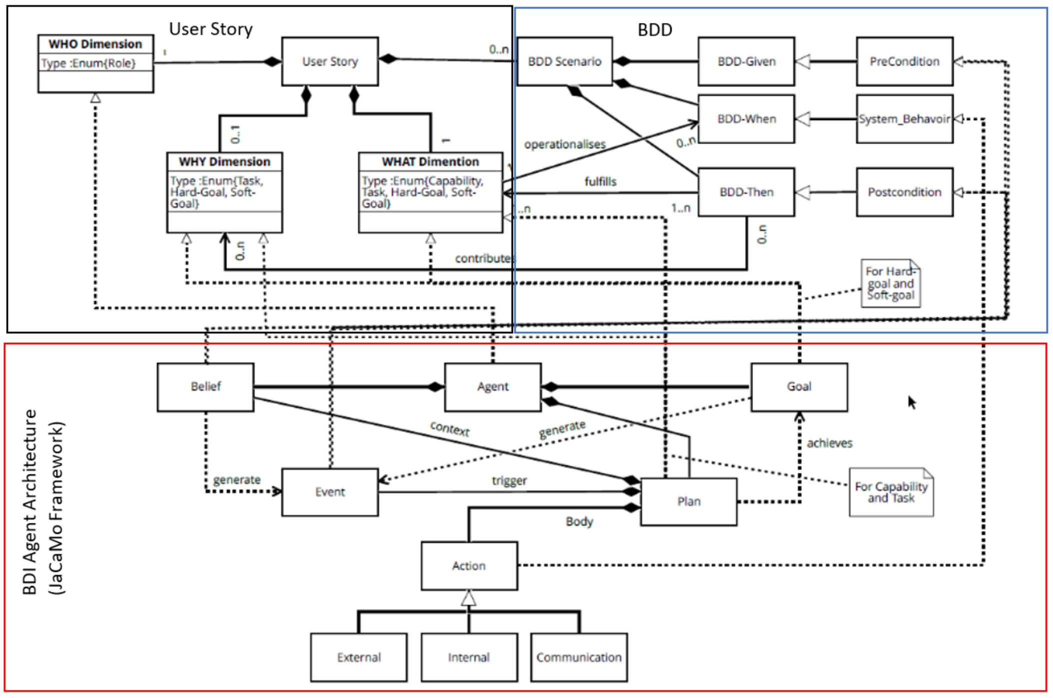

Figure 4: The mapping from user story and BDD to BDI Agent (JaCaMo framework).

A BDI agent in JaCaMo is composed of the concepts of *Belief*, *Goal*, *Plan* and *Event*. A *Belief* describes a piece of knowledge that an agent has about itself and its environment. *Events* describe stimuli, emitted by agents or automatically generated in response to which other or the same agents must take action. A Goal "represents future states of the environment that are desirable to the agent" [4]. A *Plan* describes a sequence of actions that an agent can take when an event occurs. In JaCaMo, a *Plan* is composed of an *Action* (see Fig. 4).

In the work of Wautelet et al. [8], the authors demonstrated that the concept of *Role* in the WHO-dimension of the US can be transformed into a BDI Agent. The concept of *Task* presented in both WHAT- and WHY-dimensions of a US are transformed into a *Plan*. The concept of *Capability* presented in the WHAT-dimension of a US can be transformed into an *Event* but with the JaCaMo framework the *Capability* can be mapped to a *Plan* which should, however, be composed of one *Action* only.

Hard- and soft-goals found in US can be mapped to the *Goal* part of the Agent. The BDD GIVEN- and THEN-dimensions of the BDD scenario can be mapped to the *Beliefs* and *Events* of an Agent. Lastly, the BDD WHEN-dimension can be mapped to the *Action* belonging to the *Plan* of mapped from the US it operationalizes. Fig. 4 shows the mapping of US and BDD scenarios to the Agent architecture in JaCaMo.

2.2. Object-Oriented Architecture Transformation

Mapping the elements of a US directly to elements of an OO design of an application is not straightforward: the US is formulated at a too high level to directly retrieve the necessary elements from it. The BDD scenarios corresponding to a US provide a more detailed and concrete specification. But even the mapping of BDDs to elements of an OO design is not straigthforward. More explanation could be found in [9]. Tab. 2 exemplifies the object types, attributes, and object type states that can be derived from each element of a BDD scenario. Object types and attributes can be used to defined the class diagram defining the persistent objects, while the states will be used for defining state charts per object type. Input and output services will be used to defined getters and setters for object types, and for defining user interfaces required for the interaction with the user.

Given the required expertise for deriving the elements needed for the application design from US, many researchers have already investigated possible forms of automated support for the business analys. An example is the work of [10], where US are translated to an intermediate formal language, that is subsequently used to generate UML diagrams from, making use of NLP techniques. This is by far not the only work that uses NLP to assist the analysis of US. Raharjana et al. [11] identified through an SLR a broad range of purposes of using NLP to extract some aspects of US, but conclude that it remains a significant challenge to understand a sentence's context. While the difficulties associated with NLP persist, such support can nevertheless help e.g. to improve the completeness and conciness of extracted domain models by providing traceability, as in DoMoBOT [12].

2.3. Illustration examples

Our illustration is based on a US and BDD scenarios taken from an opensource project (Simple online shop[1]). The example US is related to buying product online. The transformations to AO and OO architecture are presented in Tabs. 1 and 2 respectively.

Table 1
The illustration of transforming US and BDD scenarios to Agent architecture.

User Story and BDD	Concepts	None-BDI Agent	BDI Agent
User Story: 1: As a web user 2: I want to add products to cart 3: so that I can buy cool products	 1: User 2: AddToCart 3: BuyProduct	 1: Agent 2: SimpleBehaviour 3: CompositeBehavior	 1: Agent 2: Plan 3: Plan
BDD Scenario 1: Adding new product to cart 1: **Given** I am viewing product with Name “Super Random Product" 2: **When** I press “Add to Cart” 3: **Then** I should see “Product added to Cart” 4: **And** I should see one extra item in cart	 1: Products 2: CreateItemInCart 3: Cart 4: Cart	 1: Java Object 2: part of SimpleBehaviour 3: Java Object 4: Java Object	 1: Belief 2: Action 3: Belief 4: Belief
BDD Scenario 2: Adding extra already selected product to cart 1: **Given** I am viewing the content of my cart 2: **And** my cart already contains the product with Name “Super Random Product” 3: **When** I press “+” 4: **Then** I should see “Product added to Cart” 5: **And** I should see one extra item for “Super Random Product” in my cart	 1: Cart 2: Cart 3: AddToCart 4: Cart 5: Cart	 1: Java Object 2: Java Object 3: SimpleBehaviour 4: Java Object 5: Java Object	 1: Belief 2: Belief 3: Action 4: Belief 5: Belief

Table 2
The illustration of transforming BDD scenarios to Object-Oriented architecture.

BDD Scenario	Business Object Type referenced	Attribute Referenced	BOT state referenced	Input Service referenced	Output Service referenced
BDD Scenario 1: 1: 2: 3: 4:	 1: Product 2: Cart 3: Cart 4: ItemInCart	 1: Product Name 2: 3: 4:	 1: Product Exits 2: Cart Exits 3: 4: ItemInCart exists	 1: 2: Create ItemInCart 3: Response Success/fail 4:	 1: View Product 2: 3: 4 View Cart, View ItemInCart
BDD Scenario 2: 1: 2: 3: 4: 5:	 1: Cart 2: ItemInCart 3: 4: 5: CartItemInCart	 1: 2: 3: 4: 5:	 1: Cart Exists 2: ItemInCart Exists 3: 4: 5: Cart Exists, ItemInCart Exists	 1: 2: 3: modifyItemInCart 4: Reponse Success/Fail 5:	 1: ViewCart 2: View ItemInCart 3: 4: 5: View Cart, View ItemInCart

[1] https://github.com/kunicmarko20/Simple-Shop/tree/master/features

3. Conclusion

Our research aims at using US and BDD scenarios as a basis for building AO and OO software architecture. This paper reports the primary results of the research. We argue that some concepts presented in both US and BDD scenarios can be mapped intuitively to agent concepts. However, the mapping to object oriented models requires more in-depth analysis. This can be improved with the help of NLP which is our plan for future work as the works of [12, 13, 14, 15].

References

[1] Y. Wautelet, S. Heng, M. Kolp, I. Mirbel, Unifying and extending user story models, in: CAiSE2014, Springer, 2014, pp. 211–225.

[2] K. Tsilionis, Y. Wautelet, C. Faut, S. Heng, Unifying behavior driven development templates, in: RE2021, IEEE, 2021, pp. 454–455.

[3] Y. Wautelet, M. Kolp, Business and model-driven development of bdi multi-agent systems, Neurocomputing 182 (2016) 304–321.

[4] O. Boissier, R. H. Bordini, J. F. Hübner, A. Ricci, A. Santi, Multi-agent oriented programming with jacamo, Science of Computer Programming 78 (2013) 747–761.

[5] F. L. Bellifemine, G. Caire, D. Greenwood, Developing multi-agent systems with JADE, John Wiley & Sons, 2007.

[6] E. Yu, Agent-oriented modelling: software versus the world, in: International Workshop on Agent-Oriented Software Engineering, Springer, 2001, pp. 206–225.

[7] Y. Wautelet, S. Heng, S. Kiv, M. Kolp, User-story driven development of multi-agent systems: A process fragment for agile methods, COMLAN 50 (2017) 159–176.

[8] Y. Wautelet, S. Heng, M. Kolp, C. Scharff, Towards an agent-driven software architecture aligned with user stories., in: ICAART (2), 2016, pp. 337–345.

[9] M. Snoeck, Y. Wautelet, Agile MERODE: A Model-Driven Software Engineering Method for User-Centric and Value-Based Development, Software and Systems Modeling (accepted for publication).

[10] T. Yue, L. C. Briand, Y. Labiche, Atoucan: An automated framework to derive uml analysis models from use case models, ACM Trans. Softw. Eng. Methodol. 24 (2015).

[11] I. K. Raharjana, D. Siahaan, C. Fatichah, User stories and natural language processing: A systematic literature review, IEEE Access 9 (2021) 53811–53826.

[12] R. Saini, G. Mussbacher, J. L. Guo, J. Kienzle, Domobot: A modelling bot for automated and traceable domain modelling, in: RE2021, IEEE, 2021, pp. 428–429.

[13] F. Gilson, M. Galster, F. Georis, Generating use case scenarios from user stories, in: Proceedings of the International Conference on Software and System Processes, 2020, pp. 31–40.

[14] T. Kochbati, S. Li, S. Gérard, C. Mraidha, From user stories to models: A machine learning empowered automation., in: MODELSWARD, 2021, pp. 28–40.

[15] M. Soeken, R. Wille, R. Drechsler, Assisted behavior driven development using natural language processing, in: International Conference on Modelling Techniques and Tools for Computer Performance Evaluation, Springer, 2012, pp. 269–287.