

# Enabling Value Co-Creation in Customer Journeys with VIVA

Iván S. Razo-Zapata<sup>1</sup>, Eng K. Chew<sup>2</sup>, Qin Ma<sup>3</sup>, Loïc Gammaitoni<sup>3</sup>, and Henderik A. Proper<sup>1</sup>

<sup>1</sup> Luxembourg Institute of Science and Technology, L-4362 Esch-Sur-Alzette, Luxembourg

E-mail: {ivan.razo-zapata,erik.proper}@list.lu

<sup>2</sup> University of Technology Sydney, Australia

E-mail: eng.chew@uts.edu.au

<sup>3</sup> University of Luxembourg

## Abstract

We present a Visual language to design Value (VIVA) co-creation (VCC) for a given business from a customer perspective. VIVA aims to clearly map the high-level foundations of VCC onto generic and specific requirements to design and monitor VCC. VIVA's main concepts and relationships are inspired by ideas in business modelling, marketing, and service science, whereas the overall design of the language is driven by a domain specific language (DSL) engineering approach. In this paper, we validate VIVA's abstract syntax and concrete syntax using Lightning, which leads to the improvement of VIVA as well as to the definition of constraints ruling the use of VIVA. Likewise, we illustrate the use of VIVA by means of a case study within the citizen science project Watergram. Finally, we present some discussion and elaborate on future work.

## Keywords:

Business design, customer journey, value co-creation, visual language

## 1 INTRODUCTION

The contemporary transition from goods-dominant to service-dominant economies requires organizations (public or private) to reconceptualise the way they create value – a multi-actor (stakeholder) interactive process of resource integration in which the *customer* plays a dominant proactive role. Value co-creation (VCC) supports such reconceptualization endeavour since it defines the “processes and activities that underlie resource integration and incorporate different actor roles in the service ecosystem”, which ultimately lead the joint creation of benefits (i.e. value) [1].

Despite the fact that the VCC concept has been deeply studied, its highly conceptual nature, however, makes it difficult for service developers to fully operationalize the idea. As an attempt to address this gap, we have already proposed a Visual language to design Value co-creation (VIVA) [2]. Briefly, VIVA is a Visual language to design VCC for a given business from a *customer perspective*. VIVA aims to clearly map the high-level foundational requirements of VCC onto generic and specific requirements to design VCC. VIVA allows designing VCC by specifying full *customer journeys* in which an end customer and service suppliers integrate resources via three types of encounters, i.e. coordination, cooperation, and collaboration. VIVA's main concepts and relationships are inspired by ideas in business modelling, marketing, and service science, whereas the overall design of the language is driven by a domain specific language (DSL) engineering approach.

As VIVA's development is an on-going work, in this paper, we present an initial validation of VIVA's abstract syntax and concrete syntax by means of the use of the Lightning tool [3]. We also provide an exploratory work on how to monitor the overall VCC process with the aim to guarantee the fulfilment of a desired customer journey (thus VCC).

Likewise, we follow a design science research (DSR) approach as VIVA is ultimately an artefact that aims to solve “an organizational” problem [4], i.e. designing VCC from the point of view of customers. In this way, the rest of the paper is organized as follows: Section 2 presents literature review on VCC, whereas Section 3 describes the method used to build our artefact. Section 4 presents the initial design of the artefact, which is then validated using Lightning in Section 5. This is followed by an example on how to use the artefact to design a customer journey within the citizen science project Watergram (in Section 6). Afterwards, Section 7 provides discussion on main assumptions, lessons

learned, and open challenges. Finally, we present general conclusions and ideas on future work in Section 8.

## 2 LITERATURE REVIEW

Since VIVA is a visual language to design VCC, we have analysed work related not only to (1) the notion of value and (2) VCC but also to (3) the design of domain specific languages and (4) visual modelling. The former two elements (1) and (2) represent what DSR calls descriptive knowledge (i.e. what we know about the phenomena of VCC), whereas the last two (3) and (4) represent the prescriptive knowledge (i.e. how we can build up artefacts). A full description of such analysis can be found at [2]. The next paragraphs, however, briefly elaborate on some of the relevant aspects.

### 2.1 Value co-creation

Service-dominant logic (SDL) suggests the idea that value co-creation (VCC) encompasses the “processes and activities that underlie resource integration and incorporate different actor roles in the service ecosystem” [1]. Moreover, the notion of *value in use* is an important driver within the VCC process as it represents the *realized customer value* which encapsulates aspects regarding personalization, relationship, and experience [5,6].

Some work also recognizes the idea that value is a multi-dimensional and experiential phenomenon [7,8,9]. In fact, Holbrook defines customer value as an *interactive relativistic preference experience*, which manifests along four main types of value: economic, social, hedonic, and altruistic [7]. Economic value encompasses experiences that fulfill utilitarian objectives, whereas social value covers experiences that may trigger the response of others. Altruistic value deals with experiences having an intrinsic (self-justifying) nature and may impact on others too. Finally, Hedonic value refers to experiences that are appreciated for the simple pleasure they provide to you.

To actually involve the end customer in the co-creation of such experiences requires mastering the interactions between end customers and service suppliers. In this context, Ballantyne and Varey, have suggested the existence of three forms of interaction between customer and service suppliers during value creation, i.e. coordination, cooperation and collaboration [10]. Coordination is

considered an informative and persuasive interaction in which an actor tries to coerce the other, whereas cooperation is a communicational interaction that is perceived as a more equitable exchange between actors. Collaboration, in contrast, is seen as an emergent dialogical interaction in which actors learn from each other and jointly create value.

These interactions, nonetheless, actually occur within the context of a given customer journey [11]. It is within customer journeys that the overall cumulative experience ultimately influences the value being co-created by both customers and service suppliers [12]. Furthermore, successfully managing customer journeys benefits also suppliers since they can increase their revenue, improve employee satisfaction, and distinguish much better in the market [12].

## 2.2 Design of Domain Specific Languages

As explained in [2], VIVA has been designed as a domain specific modelling language (DSML). There are a few methods to design DSMLs. Karagiannis has proposed an agile modelling method engineering that covers the *creation, design, formalisation, development, and deployment/validation* of a DSL [13]. In a similar way, Frank has designed a method that encompasses seven steps: *clarification of scope and purpose, analysis of generic requirements, analysis of specific requirements, language specification, design of graphical notation, development of modelling tool, and evaluation and refinement* [14].

To tackle the design of graphical notation, Frank suggests following Moody's principles, which deal with: semiotic clarity, perceptual discriminability, semantic transparency, complexity management, cognitive integration, visual expressiveness, dual coding, graphic economy, and cognitive fit [15]. There are also tools to help language designers with the agile validation of DSMLs. One of them is the Lightning workbench that allows formal specification and agile validation of DSMLs [3]. It has been applied to validate DSMLs in various domains, from robotics [16] to business processes [17].

## 2.3 Visual Modelling

There are already different visual tools for modelling business ideas, services, and customer journeys. The well-known business model canvas (BMC) uses nine building blocks to design a business idea from the perspective of a company [18]. The e3value modelling tool provides a value-based abstraction to design business networks that are composed of several companies working together in a service delivery process [19]. Both, BMC and e3value, however do not focus on the customer perspective since they take the point of view of a single company (BMC) or a network of companies (e3value). BMC, however, has been recently extended with the so-called Value Proposition Canvas (VPC) with the aim to look closer at the interaction between a customer and a company [20]. There have been also recent efforts on providing support for modelling customer journeys [21, 22]. They all, however, do not differentiate the various forms of interaction taking place between customers and providers. For example, as highlighted by Ballantyne and Varey [10], coordination, cooperation and collaboration interactions are actually key elements within VCC.

# 3 METHOD

## 3.1 Design Science Research Approach

We follow a design science research (DSR) approach to build up VIVA [2,4]. Section 3.2 presents the main method applied during the design of VIVA, whereas Section 4 and Section 5 respectively present the main artefact and its validation with lightning.

## 3.2 VIVA as a Domain Specific Modelling Language

To design VIVA's main elements, we have followed Frank's method [14]. The first three steps are covered in Section 3.2.1, 3.2.2, and 3.2.3. The last steps are covered in the remaining sections of the paper. Section 4 introduces the language specification and the graphical notation. To improve the language specification and the design of the graphical notation, we have applied Lightning in Section 5. The evaluation and refinement is performed in Section 6 via a case study. Note that we still do not cover the development of the modelling tool, as it requires actual software implementation, which is outside the scope of this paper.

### 3.2.1 Clarification of scope and purpose.

VIVA's main purpose is to enable business users to design and model their desired value co-creation processes for a given business or service context [2]. Unlike BMC and e3value, VIVA focuses on customers and how they interact with service providers by supporting the design of customer journeys.

### 3.2.2 Analysis of generic requirements.

As explained in [2], the development of VIVA must satisfy four generic requirements: *VCC design (GR1)*: VIVA must enable business users to design VCC for a given business or service context (e.g. a travel journey), *Communication (GR2)*: VIVA should support communicating ideas among stakeholders in a simple and intuitive way. *Analysis (GR3)*: VIVA should support the basic analysis of resulting designs. *Computer support (GR4)*: VIVA must be implemented as a software tool to allow designing and analysing VCC.

### 3.2.3 Analysis of specific requirements.

Based on the generic requirements, we have defined seven specific requirements [2]. *Resources (SR1)*: VIVA should be able to represent relevant resources that are integrated as part of VCC. *Forms of co-creation (SR2)*: VIVA should support describing co-ordination, co-operation and collaboration. *Beneficiary centric (SR3)*: VIVA should focus on the relationships (encounters/touchpoints) established between an *end customer* and suppliers of a service ecosystem. *Background agnostic (SR4)*: Intuitive use for different audiences (e.g. business or technical). *Visual support (SR5)*: VIVA's constructs should help users to design VCC. *Semantic support (SR6)*: Basic reasoning tasks should be supported. *Standardised representation (SR7)*: VIVA should be represented in a "formal" modelling tool.

# 4 VIVA LANGUAGE (ARTEFACT)

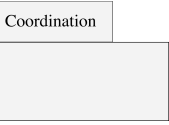



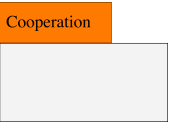


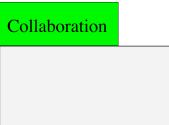





## 4.1 Language

The VIVA meta-model is presented in Figure 1. The main concepts are: *Journey, Encounter, Value, Actor, Role, and Resource*. Briefly, a Journey *contains* encounters in which two actors participate, one playing the role of customer and the second one playing the role of provider. Actors playing their respective roles bring resources to the encounter, which allows the co-creation of value [2]. Finally, we also acknowledge that abstract concepts such as encounter, value, resource, and actor can be specialized into more detailed concepts. For instance, an encounter can be either coordination, cooperation or collaboration [2].

## 4.2 Visual constructs

Table 1 presents the visual constructs that are part of VIVA and are required to design customer journeys. As one can see, we distinguish different types of encounters (cooperation, coordination, and collaboration) as well as different types of value and resource.

Table 1 VIVA Constructs

Encounter	Value	Resource	Actor
Coordination 	Economic 	Knowledge 	Human 
Cooperation 	Social 	Skill 	
Collaboration 	Hedonic 	Technology 	Machine 
	Altruistic 	Asset 	

## 5 VALIDATION WITH LIGHTNING

In Lightning, a DSML definition is composed of: (1) an abstract syntax model (ASM) defining the concepts and well-formed constraints of the language; (2) a concrete syntax model (CSM) defining how models expressed in the defined language are to be visually depicted; and (3) a semantics model (SM) defining the meaning of the language, in terms of operational semantics [17].

The specification of ASMs, CSMs, and SMs in Lightning is done in the Alloy language [23], and the transformations among them are specified in the F-Alloy language [24]. Both Alloy and F-Alloy are formal specification languages (for models and model transformations respectively) based on first-order relational logic.

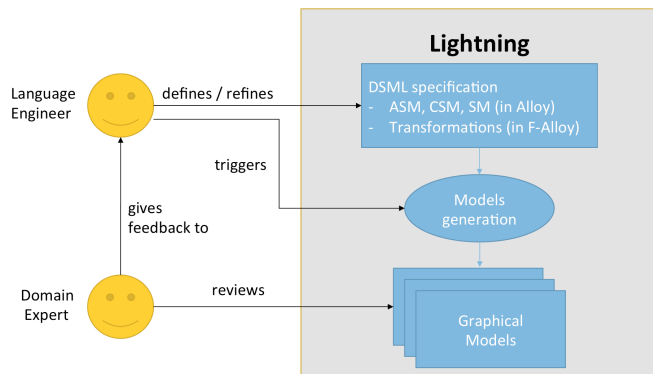


Figure 2 DSML Validation Process in Lightning

As depicted in Figure 2, after a language engineer defines a DSML in Lightning, the design of the DSML can be validated by domain experts directly in an intuitive, visual and interactive way without requiring any technical skills [25]. More specifically, instances conforming to the DSML specification or counter-example disproving some properties of the DSML are automatically generated by a so-called SATISFIABILITY (SAT) solver (the Alloy Analyzer), and rendered in a domain-specific visual notation

to be reviewed by domain experts. The language engineer then collects feedback from domain experts and refines the design of the DSML. Instances of the new version of the DSML will again be generated and rendered graphically to be validated by domain experts. Such a process can be iterated as many times as needed until the designed DSML meets the expectation of the domain experts.

In the following, we will validate the VIVA language with Lightning. We start by validating the abstract syntax of VIVA (the metamodel depicted in Figure 1). After several iterations, when instances generated from the abstract syntax all satisfy the expectation of the domain expert, we move on to investigating alternatives of concrete syntax of VIVA.

### 5.1 Validating VIVA abstract syntax

#### 5.1.1 Define VIVA ASM

The Alloy model defining the ASM of VIVA is as follows:

```
module VIVA/AbstractSyntax/ASM
```

```
sig Journey {
    description: one disj String,
    // one means the multiplicity of description is [1..1]
    // disj means journeys have distinct descriptions
    contains: seq Encounter
    // seq means encounters are ordered
}

some contains.elms
// some means the multiplicity of contains is [1..*]

abstract sig Encounter {
    description: disj String,
    hasCustomer: one Customer,
    hasProvider: one Provider,
    coCreatesValue: some Value
}

one this.~contains
// ~.contains refers to the inverse of contains
// which is the isContainedIn reference (cf. Figure 1)
// one means the multiplicity of isContainedIn is [1..1]
#coCreatesValue>1 and #coCreatesValue<9
// this fact restrains the multiplicity of contains to [2..8]
}
```

```
sig Coordination, Cooperation, Collaboration extends Encounter {}
// Three subclasses of Encounter
```

```
abstract sig Actor {
    playsRole: some Role,
    name: one disj String
}
```

```
sig Human, Machine extends Actor {}
// Two subclasses of Actor
```

```
abstract sig Role {
    brings: some Resource,
    benefitsFrom: some Value
}
```

```

#brings<=4
// this fact restrains the multiplicity of brings to [1..4]
#benefitsFrom<=4
// this fact restrains the multiplicity of benefitsFrom to [1..4]
one this.~playsRole
// the multiplicity of playedByActor,
// which is the inverse of playsRole
// is restrained to [1..1]
}

sig Customer extends Role{}{
    one this.~hasCustomer
    // the multiplicity of participatesInEncounter,
    // which is the inverse of hasCustomer
    // is restrained to [1..1]
}

sig Provider extends Role{}{
    one this.~hasProvider
    // the multiplicity of participatesInEncounter,
    // which is the inverse of hasProvider
    // is restrained to [1..1]
}

abstract sig Resource {
    description: one disj String,
}

    some this.~brings
    // the multiplicity of broughtBy,
    // which is the inverse of brings
    // is restrained to [1..*]
}

abstract sig Operant extends Resource{}
sig Knowledge, Skill extends Operant{}
abstract sig Operand extends Resource{}
sig Technology, Asset extends Operand{}
abstract sig Value{
    description: one disj String
}

    one this.~benefitsFrom
    // the multiplicity of providesBenefitTo,
    // which is the inverse of benefitsFrom
    // is restrained to [1..1]
    one this.~coCreatesValue
    // the multiplicity of coCreatedBy,
    // which is the inverse of coCreatesValue
    // is restrained to [1..1]
}

sig EconomicValue, SocialValue, HedonicValue, AltruisticValue extends Value{}

```

This ASM is a straightforward translation to Alloy of the class diagram given in Figure 1. More specifically, (abstract) classes are defined as Alloy (abstract) **signatures**, and structural features (i.e., references and attributes) are defined as Alloy **fields**. A field “f: S2” defined in signature S1 represents a relation between the two signatures S1 and S2. For example, in the Encounter signature above, a field “hasCustomer: Customer” is defined to represent the “hasCustomer” reference from class Encounter to class Customer as specified by the metamodel in Figure 1.

It is worth noticing how opposite references are handled in Lightning. According to the metamodel specification in Figure 1, the opposite of “hasCustomer” is defined by another reference called “participatesInEncounter”. In Lightning, thanks to the inverse operator “~” of relations provided by Alloy, we only need to define one relation for the reference “hasCustomer”, and we can refer to the opposite reference “participatesInEncounter” simply by “~hasCustomer”.

### 5.1.2 Generate instances of VIVA ASM

After specifying the ASM of VIVA, Lightning executes this specification in the Alloy Analyzer to exhaustively generate all instances conforming to the ASM within a pre-defined (relatively small) scope. The effectiveness of this approach relies on the so-called “small scope hypothesis”, which states that “errors in specifications can usually be demonstrated with small counterexamples” [26].

Generated instances are supposed to be presented to domain experts for validation one after another in order for them to detect potential anomalies. As in most cases, domain experts do not necessarily possess the skills to interpret instances that are generated directly by the Alloy Analyzer. To cope with this, Lightning allows us to give an intuitive domain specific visualization to the generated ASM instances. This is done by specifying a model transformation in Lightning from the VIVA ASM to a visual modelling language (VLM). VLM is a standard Alloy module defined in Lightning that offers a set of common graphical concepts such as shapes (e.g., rectangles and ellipses), colours, layouts, and connectors (e.g., arrows or lines) that allow to connect shapes. With the help of the VIVA-ASM-to-VLM transformation, a VLM instance (following the visual constructs defined in Section 4.2) will be automatically generated for each generated VIVA ASM instance, and Lightning can parse and graphically render these VLM instances. It is interesting to note that the ASM to VLM model transformation is expressed in the F-Alloy language, a language only supported by Lightning allowing the concise specification in Alloy of efficiently computable model transformations.

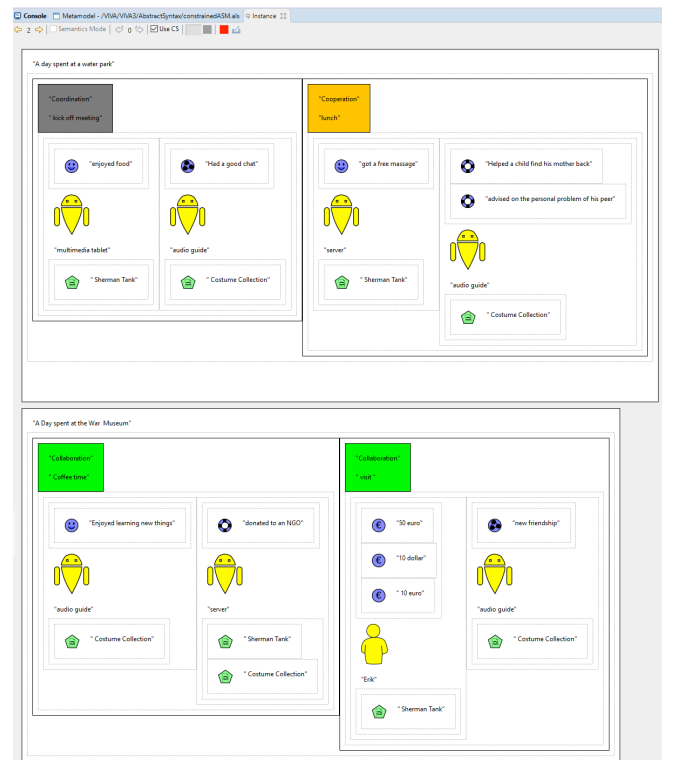


Figure 3 An instance conforming to VIVA ASM and rendered visually using the notations defined in Section 4.2

We show in Figure 3 one of the instances generated by Lightning for VIVA ASM. This instance does not correspond to any real-life VCC scenario because it is automatically generated with random descriptions. However, it is an interesting instance because it reveals several inaccuracies in the VIVA ASM specification.

- 1) There are two journeys “A Day spent at a Water Park” and “A Day spent at the War Museum” in this instance. However, according to domain knowledge, only one journey can exist in the context of a VCC scenario (C1).
- 2) The second journey starts directly with a collaboration encounter “Coffee time”. However, according to domain knowledge, each journey should start with a coordination encounter (C2), and a collaboration encounter can only take place after a cooperation encounter (C3).
- 3) The first journey ends with a cooperation encounter “lunch”. However, according to domain knowledge, to reach the goal of VCC, each journey should go through at least one collaboration encounter (C4).
- 4) In the first journey, the actor “audio guide” plays the role of provider in the first encounter and the role of customer in the second encounter. However, according to domain knowledge, in the context of a journey, an actor cannot be both the customer and the provider (C5).
- 5) In the second journey, the actor “audio guide” plays the role of customer in the first encounter and the actor “Erik” plays the role of customer in the second encounter. However, according to domain knowledge, in the context of a journey, the role of customer must be played by the same actor across all the encounters (C6).
- 6) There are three encounters (the two encounters of the first journey and the first encounter of the second journey) in which the role of customer is played by a machine actor. However, according to domain knowledge, the role of customer can only be played by human actors (C7).
- 7) In the second encounter “visit” of the second journey “A Day spent at the War Museum”, the customer “Erik” benefits from three times economic values (“50 euro”, “10 dollar”, and “10 euro” respectively). However, according to domain knowledge, values benefiting a role should all be of different value types (C8).
- 8) In the first encounter “Coffee time” of the second journey, the provider “server” brings twice asset resources (“Sherman Tank” and “Costume Collection”). However, according to domain knowledge, resources brought by a role should all be of different resource types (C9).

If the language engineer specifies such domain knowledge in the form of explicit constraints, those mistakes can easily be avoided. However, they were somehow overlooked because of the knowledge gap between domain experts and the language engineer. More specifically, just like the domain experts have no idea about the Alloy syntax, the language engineer has no pre-existing knowledge of the value co-creation domain either. Lightning overcomes the first kind of knowledge gap by enriching the automatically generated ASM instance with a domain-specific visualization. And the iterative “specification, generation and validation” process of Lightning helps to reduce exactly the second kind of knowledge gap. There is no other effective means for the domain experts to realize the imprecisions in the language specification except for letting them witness faulty example instances.

### 5.1.3 Refine VIVA ASM with missing constraints

Following is the refined VIVA ASM. It imports the previous VIVA ASM and specifies additional constraints (defined as Alloy “facts”)

to capture the missing domain knowledge that is assumed by domain experts.

```

module VIVA/AbstractSyntax/constrainedASM
open VIVA/AbstractSyntax/ASM

fact C1 { // For each VIVA model, there is EXACTLY ONE journey
    one Journey
}
fact C2 { // A journey should start with a Coordination encounter
    Journey.contains[0] in Coordination
}
fact C3 { // A Collaboration cannot take place without a cooperation
    encounter having taken place already.
    all c: Collaboration |
        some x: Cooperation |
            Journey.contains.idxOf[x] < Journey.contains.idxOf[c]
}
fact C4 { // A Journey should have at least one Collaboration encounter
    Journey.contains elems & Collaboration != none
}
fact C5 { // An actor CANNOT play the role of providers and consumer in
    the same journey
    no Encounter.hasConsumer.~playsRole
        & Encounter.hasProvider.~playsRole
}
fact C6 { //The consumer MUST BE played by the same actor in all
    encounters
    one Encounter.hasConsumer.~playsRole
}

fact C7 { // The role of consumer CAN ONLY BE played by a human actor
    Encounter.hasConsumer.~playsRole in Human
}
fact C8 { // The values providing benefits to a role MUST BE from different
    value types
    all r: Role {
        lone r.benefitsFrom & EconomicValue
        lone r.benefitsFrom & SocialValue
        lone r.benefitsFrom & HedonicValue
        lone r.benefitsFrom & AltruisticValue
    }
}
fact C9 { // The resources brought by a role MUST BE from different
    resource types
    all r: Role {
        lone r.brings & Skill
        lone r.brings & Knowledge
        lone r.brings & Technology
        lone r.brings & Asset
    }
}

```

Instances of the refined VIVA ASM are again exhaustively generated and reviewed by the domain experts within a given scope. This time, no errors are observed any more, and the domain experts accept the validity of the abstract syntax.

## 5.2 Investigating alternative concrete syntaxes

The modularity of the language specification in Lightning allows to seamlessly try out different concrete syntax for the same language. Indeed, to do so, it suffices to simply define alternative VIVA-



ASM-to-VLM transformations. In this paper, we demonstrate two ways of presenting VIVA customer journeys: the first alternative follows the de facto standard way of presenting customer journeys in the form of tables [22], while the second alternative adopts a graph-like appearance. For illustration purpose, we present the same watergram project example discussed in Section 6 in both formats in Figure 4 and Figure 5 respectively.

## 6 EXAMPLE

To illustrate the use of VIVA to enable VCC, we have applied it to the design of a customer journey within an on-going project. The watergram project aims to raise awareness on water quality in Luxembourg by exploring an approach based on so-called citizen science [27]. Watergram relies on citizens (mostly young students enrolled in high-school) working together with scientist and science communicators/facilitators to increase the amount and quality of data on water quality in Luxembourg (see also <http://waterhackers.lu/>). Moreover, one of the main elements of the watergram project is a digital platform that should allow students to upload water samples with information about the quality of water measured in such samples (i.e. pH, oxygen, and turbidity). VIVA is used in this context to model the journey travelled by students that allows co-creating value at different stages.

As depicted in Figure 4 and Figure 5 (an alternative concrete syntax generated with Lightning), the journey starts with the students getting trained on basic concepts regarding the water cycle (first encounter) and the different analyses that can be done to a water sample (second encounter). Both encounters are modelled as coordination since the laboratory scientist (in the first encounter) and the laboratory technician (in the second encounter) are actually leading the value creation process. Likewise, both encounters indicate the resources each actor brings (e.g. the students “bring” their basic understanding of water properties) as well as the value they expect to create (e.g. the students expect to learn about the water cycle). Note that acquiring knowledge is modelled as an economic value because it actually fulfils a utilitarian purpose, i.e. learning about something.

Later on, the journey evolves into a more cooperative interaction since the students and the workshop facilitator work together building so-called Do It Yourself (DIY) tools. The students bring to these encounters the knowledge that was acquired in the first two encounters.

The last two encounters are actually real collaborations between the students, the web developer, and the water scientist. Within the co-design of the watergram platform, the group of students not only learn about co-designing platforms and have fun with it but also provide important user requirements to the web developers, i.e. both sides really learn from each other. Likewise, during the collection, analysis and upload of water samples, the group of students and the water scientist both learn from each other. On the one hand, students learn about collecting, analysing and uploading samples in a natural setting. On the other hand, the water scientist learns more about water quality in Luxembourg as well as from students’ feedback.

## 7 DISCUSSION

Although VIVA has been previously evaluated in [2], the validation performed with Lightning has provided relevant insights, which will be used to improve VIVA’s design as well as the final development in a software tool. In this sense, the discovery of unseen constraints is one of the most relevant insights. This also confirms the added value of using Lightning.

By using VIVA in the Watergram project, we also discovered two relevant findings. First, we observe that VCC actually involves a knowledge creation process in which providers and customers have the opportunity to actually learn from each other. Second, while still giving some room to creativity, VIVA forces designers to precisely define encounters and to think about the value(s) being co-created as well as the resources that are needed. This property seems important to better structure customer journeys that can ultimately achieve VCC.

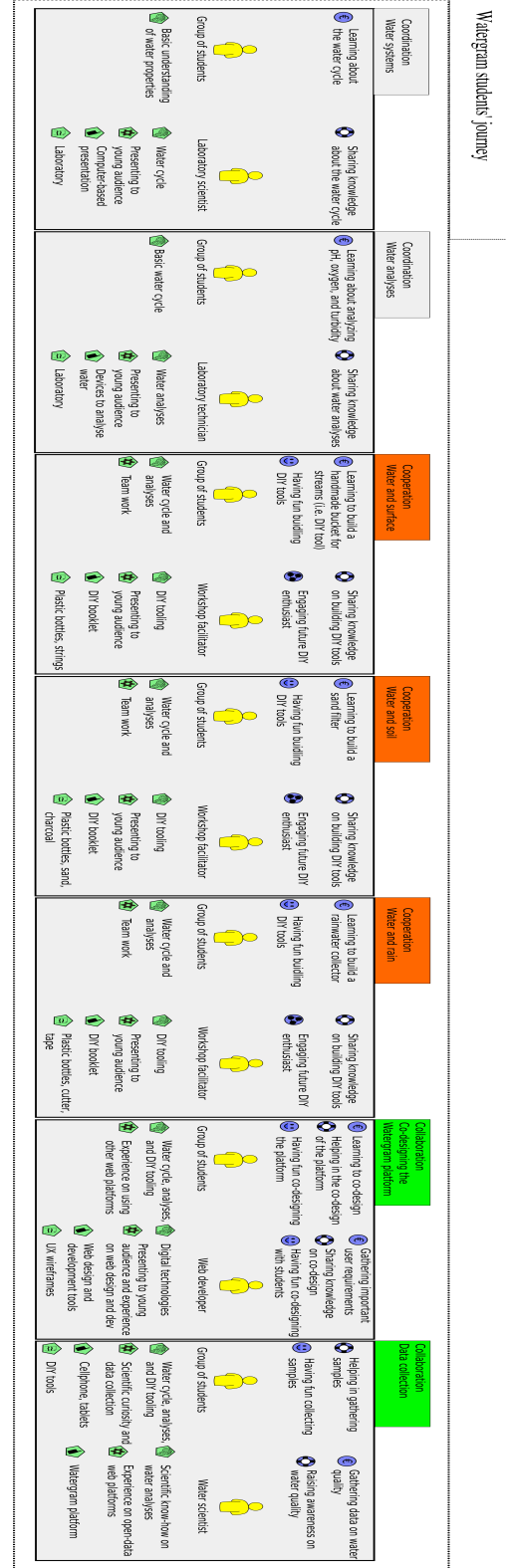


Figure 4. Watergram student's journey

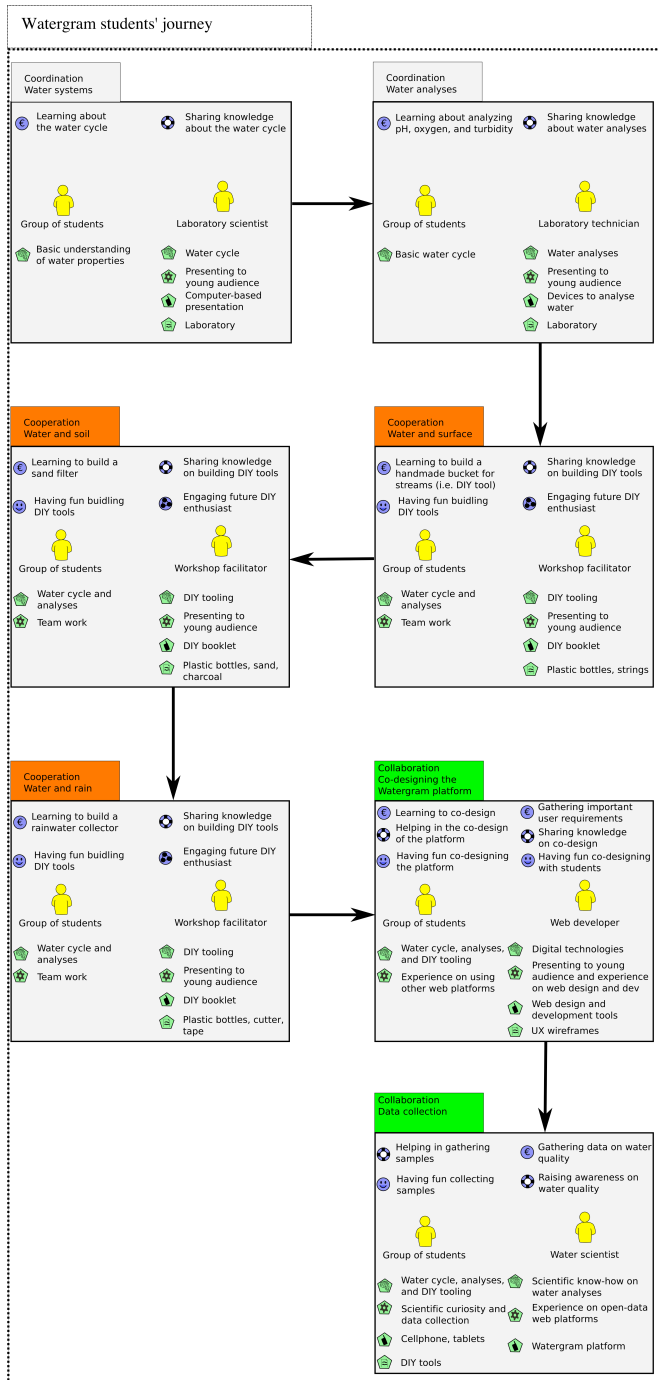


Figure 5. Watergram student's journey (alternative concrete syntax)

## 8 CONCLUSIONS AND FUTURE WORK

VIVA is a promising tool that has been also previously evaluated by a group of potential users (see [2]). The overall tool, however, still requires some improvement. In this way, the next steps will focus on validating VIVA within other real-world case studies as well as on the development in a software tool. To achieve the first step, we plan to contact business designers working on projects that require innovation via the design of new services. Regarding the second step, we will continue iterating with Lightning to later on achieve a mature description to be implemented in modelling frameworks such as ADOxx (<https://www.adoxx.org/live/home>).

## 9 REFERENCES

- [1] R. F. Lusch and S. Nambisan, "Service innovation: A service-dominant logic perspective." *Mis Quarterly*, vol. 39, no. 1, pp. 155–175, 2015.
- [2] I. S. Razo-Zapata, E. K. Chew, and E. Proper, "VIVA: A Visual Language to Design Value Co-creation" in 20th IEEE International Conference on Business Informatics, 2018.
- [3] Gammaitoni, L., Kelsen, P., & Glodt, C. (2015, October). Designing languages using lightning. In Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering (pp. 77-82). ACM.
- [4] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of Management Information Systems*, vol. 24, no. 3, pp. 45–77, 2007. [Online]. Available: <http://www.tandfonline.com/doi/abs/10.2753/MIS0742-122240302>
- [5] A. F. Payne, K. Storbacka, and P. Frow, "Managing the co-creation of value," *Journal of the Academy of Marketing Science*, vol. 36, no. 1, pp. 83–96, 2008. [Online]. Available: <http://dx.doi.org/10.1007/s11747-007-0070-0>
- [6] K. R. Ranjan and S. Read, "Value co-creation: concept and measurement," *Journal of the Academy of Marketing Science*, pp. 1–26, 2014. [Online]. Available: <http://dx.doi.org/10.1007/s11747-014-0397-2>
- [7] M. B. Holbrook, "Consumption experience, customer value, and subjective personal introspection: An illustrative photographic essay," *Journal of Business Research*, vol. 59, no. 6, pp. 714 – 725, 2006.
- [8] Martina G. Gallarza, Francisco Arteaga, Giacomo Del Chiappa, Irene Gil-Saura, Morris B. Holbrook, (2017) "A multidimensional service-value scale based on Holbrook's typology of customer value: Bridging the gap between the concept and its measurement", *Journal of Service Management*, Vol. 28 Issue: 4, pp.724-762, <https://doi.org/10.1108/JOSM-06-2016-0166>
- [9] K. R. Ranjan and S. Read, "Value co-creation: concept and measurement," *Journal of the Academy of Marketing Science*, pp. 1–26, 2014. [Online]. Available: <http://dx.doi.org/10.1007/s11747-014-0397-2>
- [10] D. Ballantyne and R.J. Varey, "Creating value-in-use through marketing interaction: the exchange logic of relating, communicating and knowing," *Marketing theory*, vol. 6, no. 3, pp. 335–348, 2006.
- [11] A. F. Payne, K. Storbacka, and P. Frow, "Managing the co-creation of value," *Journal of the Academy of Marketing Science*, vol. 36, no. 1, pp. 83–96, 2008. [Online]. Available: <http://dx.doi.org/10.1007/s11747-007-0070-0>
- [12] A. Rawson, E. Duncan, and C. Jones, "The truth about customer experience," *Harvard Business Review*, vol. 91, no. 9, pp. 90–98, 2013.
- [13] D. Karagiannis, "Agile modeling method engineering," in Proceedings of the 19th Panhellenic Conference on Informatics, ser. PCI '15. New York, NY, USA: ACM, 2015, pp. 5–10. [Online]. Available: <http://doi.acm.org/10.1145/2801948.2802040>
- [14] U. Frank, "Domain-specific modeling languages: requirements analysis and design guidelines," in Domain Engineering. Springer, 2013, pp. 133–157.
- [15] D. Moody, "The physics of notations: Toward a scientific basis for constructing visual notations in software engineering," *Software Engineering, IEEE Transactions on*, vol. 35, no. 6, pp. 756–779, Nov 2009.
- [16] Gammaitoni, L., & Hochgeschwender, N. (2016). RPSL meets lightning: A model-based approach to design space exploration of robot perception systems. RPSL meets lightning: A model-based approach to design space exploration of robot perception systems.

[17] Gammaitoni, L., Kelsen, P., & Mathey, F. (2014). Verifying modelling languages using lightning: a case study. MoDeVVA 2014: Model-Driven Engineering, Verification and Validation, 19-28.

[18] A. Osterwalder, "The business model ontology - a proposition in a design science approach," Ph.D. dissertation, University of Lausanne, Ecole des Hautes Etudes Commerciales HEC, 2004.

[19] J. Gordijn and J. Akkermans, "Value-based requirements engineering: exploring innovative e-commerce ideas," Requirements Engineering, vol. 8, pp. 114-134, 2003, 10.1007/s00766-003-0169-x. [Online]. Available: <http://dx.doi.org/10.1007/s00766-003-0169-x>

[20] B. model generation, "Value proposition canvas," <https://strategyzer.com/canvas/value-proposition-canvas>, [Online; accessed 01-February-2018].

[21] R. Halvorsrud, E. Lee, I. M. Haugstveit, and A. Følstad, "Components of a visual language for service design," in ServDes. 2014 Service Future; Proceedings of the fourth Service Design and Service Innovation Conference; Lancaster University; United Kingdom; 9-11 April 2014, no. 099. Linköping University Electronic Press, 2014, pp. 291-300.

[22] Kalbach, J. (2016). Mapping experiences: A complete guide to creating value through journeys, blueprints, and diagrams. " O'Reilly Media, Inc."

[23] Jackson, D. (2006). Software abstractions (Vol. 2). Cambridge: MIT press.

[24] Gammaitoni, L., & Kelsen, P. (2017). F-Alloy: a relational model transformation language based on Alloy. Software & Systems Modeling, 1-35.

[25] Gammaitoni, L., Kelsen, P., & Ma, Q. (2016, July). Agile validation of higher order transformations using F-Alloy. In Theoretical Aspects of Software Engineering (TASE), 2016 10th International Symposium on (pp. 125-131). IEEE.

[26] Jackson, D. (1998). Boolean compilation of relational specifications. Technical Report. Massachusetts Institute of Technology Cambridge, MA, USA.

[27] Franzoni, C., & Sauermann, H. (2014). Crowd science: The organization of scientific research in open collaborative projects. Research policy, 43(1), 1-20.

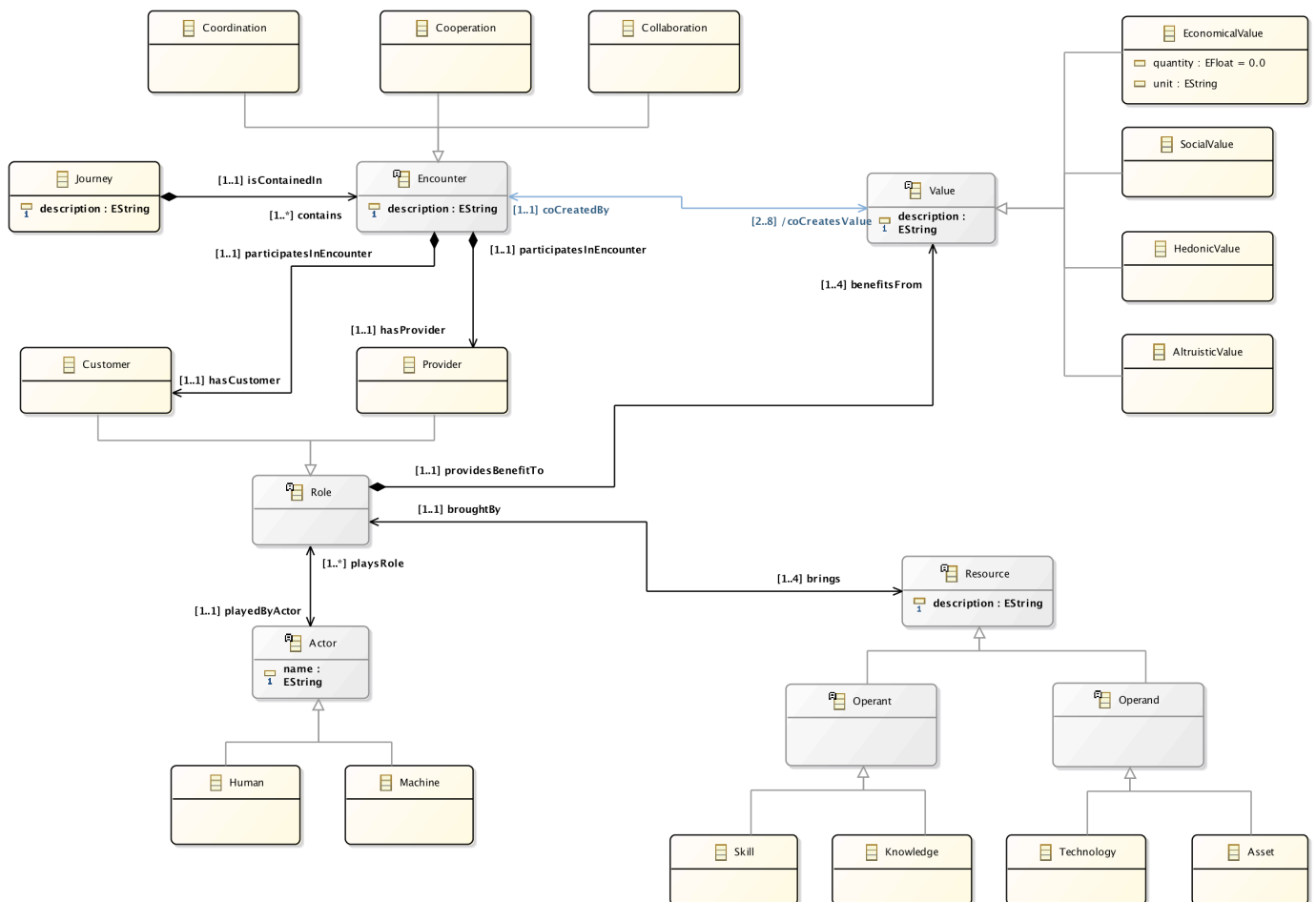


Figure1 VIVA metamodel