SACJ 32(1) December 2019
Research Article

A functional ontology for information systems

Yusuf M Motara[a], Karl van der Schyff[b]

[a] Department of Computer Science, Rhodes University, South Africa
[b] Department of Information Systems, Rhodes University, South Africa

ABSTRACT
The ontology of information systems — the way in which knowledge claims, and thus theories, are conceptualised and represented — is of particular importance in the information systems field, due to its reliance on relations between entities. This work proposes, demonstrates, and evaluates an alternative ontology for theory description which is arguably more powerful and more expressive than the dominant ontological model.

Keywords: ontology, theory building, functional design

Categories: • **Applied computing ∼ Enterprise ontologies, taxonomies and vocabularies** • Software and its engineering ∼ System modeling languages

Email:
Yusuf M Motara y.motara@ru.ac.za (CORRESPONDING),
Karl van der Schyff k.vanderschyff@ru.ac.za

Article history:
Received: 15 Mar 2019
Accepted: 16 Oct 2019
Available online: 20 Dec 2019

1 INTRODUCTION

Information systems research strives to be both globally applicable and locally relevant. It cannot, however, be globally applied unless it can be successfully elevated from its local context. Such elevation can only be achieved through the use of theories to generalise locally relevant research.

In a reflection of his thirty-year history in the field, Weber makes the claim that *"the development of new theory and the refinement of existing theories have been relatively neglected features of research within the information systems discipline"* (2016) and proposes a solution based on the philosophical work of Bunge (1977). The claim can be debated — after all, *"relatively neglected"* is in the eye of the beholder — but this work does not attempt to debate it; instead, taking it as *prima facie* true, it proposes an alternative ontology that may be more useful for describing the problem domains of information systems.

It is more specifically argued that overarching theories are not applied in specific contexts because *"no single theory is likely to accommodate [a] diversity of objectives"* (Heffernan, Lin, & Thomson, 2016); indeed, there are a large number of theories that could be applied and often are, though

Motara, Y. M., and Van der Schyff, K. (2019). A functional ontology for information systems. *South African Computer Journal 32*(1), 57–76. https://doi.org/10.18489/sacj.v31i2.691

in various restricted contexts. This is true; however, is the problem the theories themselves, or the way in which the theories are represented? The contribution of this paper is an ontology[1] that may make it easier to refine, extend, and develop an overarching theory — or, at least, to integrate one theory with another. A significant part of the value of this, and one of the reasons that the work was undertaken, is to provide a genuine alternative to the mainstream way of conceptualising theories in the Information Systems field. Even if such an alternative is not used, the fact of its existence may spur others to question the deepest fundamentals of the field and thus stimulate important research.

The structure of this paper is as follows. It begins with a background section to highlight the importance of theory, summarise Gregor's taxonomy of information systems theory (2006), and describe Weber's framework for evaluating the strength of a theory (2012). An ontology based on type theory and inspired by functional programming is then described. The utility of such an ontology is demonstrated by modeling the Theory of Planned Behaviour (Ajzen, 2005) and the Capability Approach (Sen, 1993). A discussion section that considers the benefits of the proposed ontology, often with reference to the demonstrated models, follows. An ontology is a theory that can be used to describe theories, and as such, Weber's framework is applicable to it. The framework is therefore used to evaluate the strength of the proposed ontology, and to compare it to the more dominant Bunge-Wand-Weber ontology (Wand & Weber, 1993; Weber, 2012). The paper ends with some conclusions that can be drawn.

2 BACKGROUND

The word "ontology" has both a modern technical meaning, as exemplified by the Web Ontology Language specification, and a more ancient (but still philosophically current) meaning that defines it as being the study of "being" itself. What exists? How do we relate things which exist to each other? What entities or relationships are fundamental? All of these are ontological questions, and any system which claims to answer them is therefore an ontology.

Although the way in which the very concept of a theory is understood and conceptualized differs in the field, the importance of theory to a field cannot be understated. The work of Maton (2011) uses legitimation code theory to argue that a discipline *qua* discipline cannot exist without theories to ground it and bind it within an epistemological framework; Neumann, Parry, and Becher (2002) point out that theories may be used for different purposes in different disciplines (hard/soft, pure/applied), but are nevertheless universally present; and Gregor (2006) engages in a deep examination of the different kinds of theories that exist within the information systems field.

Theories are not merely abstractions within the field: they have important practical implications. Gašević, Guizzardi, Taveter, and Wagner provide a partial list of these in the introduction to a special issue of *Information Systems* (2010). There is also some important practical disciplinary work that would be impossible to analyse or understand without theoretical backing. For example, zur Muehlen and Indulska (2010) compare several process-related languages and modeling tools using representational analysis; without the theoretical foundations of representational analysis,

[1]A way to conceptualise and represent knowledge claims, and thus theories

such work would be literally inconceivable.

The work of Gregor specifically focuses on what they describe as *"Structural or ontological questions"* in relation to information systems theory (2006, p. 612):

> What is theory? How is this term understood in the discipline? Of what is theory composed? What forms do contributions to knowledge take? What types of claims or statements can be made? What types of questions are addressed?

The term *"theory"* is defined as referring to *"abstract entities that aim to describe, explain, and enhance understanding of the world and, in some cases, provide predictions of what will happen in the future and to give a basis for intervention and action"* (2006, p. 616). It is understood, of course, that each researcher may have their own understanding of a theory. However, for a theory to be useful as an abstraction in the field, there must be sufficient consensus within the field to minimise ambiguity.

At the highest level of abstraction, there is *"meta-theory"* which *"provides a way of thinking about other theories"* (2006, p. 616). This conception of "meta-theory" is congruent with the already-expressed philosophical meaning of "ontology" and Gregor's own understanding of ontology as "a language for talking about the nature and components of theory" (2006, p. 612). Subordinate to this are actual theories which exist for the following purposes (abridged from (2006, p. 620)):

I **Analysis**. The theory does not extend beyond analysis and description.

II **Explanation**. The theory provides explanations but does not aim to predict with any precision.

III **Prediction**. The theory provides predictions and has testable hypotheses but does not have well-developed justificatory causal explanations.

IV **Explanation and prediction**. Provides predictions and has both testable propositions and causal explanations.

V **Design and action**. The theory gives explicit prescriptions for constructing an artifact.

It could be argued that the meta-theory from which the majority of information systems theories take their roots is a simplified version of Plato's theory of Forms (Fine, 2003). Fundamental to the Platonic view is the concept of objects which have some ideal form, the properties that these forms have, and—under some interpretations (Castañeda, 1972)—the relations between these objects. Indeed, the Object Management Group (OMG), whose specifications are core to much of the practical work in information systems, premises the most successful of its specifications (including UML, CORBA, SYSML, BPMN, and MOF) on such a Platonic view. Such specifications are Type-I, Type-II and/or Type-V theories in the above taxonomy, and may even be considered Type-III and/or Type-IV theories when backed by a suitably rigorous contextual analysis and discussion.

Given the above, lines may be drawn between ontology, theory, knowledge, and language. An ontology expresses what may exist and the relationships which may exist between things that exist. A theory constructs abstract entities within that ontological system to describe, explain, predict, and so on; philosophically speaking, a theory is more concerned with epistemology than ontology and, thus, the justification of claims within a domain. Knowledge is a domain-specific, concrete exemplar within a theory. Language is notation which may be used for the representation of knowledge, ontology, or theory.

2.1 Weber's framework

Weber's framework for the evaluation of theories is important to understand for two reasons: it provides a way, familiar to information systems practitioners, for the proposed ontology to be evaluated; and it provides a baseline that can be compared to the proposed onology. In the interest of clarity of exposition, citations in this section have been kept to a minimum; interested readers are encouraged to read the relevant works (Wand & Weber, 1993; Weber, 2012) in full. Weber takes a particular view of theory that they separate from *"typologies"* and *"models"* (2012, p. 4-5), but in a work that aims to be of general utility, the arguments put forward by Gregor (2006, p. 632-633) in favour of comprehensiveness are considered to be superior. Quoting Weick (1995, p. 386) as they do,

> We would like writers to feel free to use theory whenever they are theorizing. Modesty is all very well, but leaning over too far backward removes a good word from currency.

Weber proceeds by setting up an ontology for theories and then using that ontology to evaluate the strength of any particular theory. A very similar ontology appears in a previous collaborative work of Weber (1993), where it is called the Bunge-Wand-Weber ontology; this name has been reused to refer to the updated ontology for ease of reference, although the later paper does not make use of this term. The ontology itself falls within the abovementioned meta-theory of Forms. A paragraph suffices to summarise it, with capitalised words in **bold** being the names of constructs in Weber's ontology (2012, p. 3-4).

The world consists of conceptual and concrete **Things**, which may interact with each other, and which may contain other Things and therefore become **Composite Things**. **Properties** are not Things, but are descriptive attributes of Things; all Things have Properties. Common Properties of Things may be generalised into a **Class**. The way that a Property is perceived at a point in time is called an **Attribute**, and since Things are comprised of Properties, they may be described as a vector of their attributes (called their **State**). The change of State is caused by an **Event**, and the ordered sequence of its States over time is called its **History**. States and Events may both be **Lawful** or not, where *"lawful"* relates to either *"natural or human-made laws"* (2012, p. 4).

The strength of a theory is determined by how well the theory meets certain criteria (Weber, 2012, p. 4). These criteria are defined from two complementary perspectives: *parts* and *whole*. The parts of a theory "circumscribe the *boundary* or *domain* of the theory—that is, the phenomena it is intended to cover" (2012, p. 6) (emphasis in original). Parts are evaluated according to the following criteria (2012, p. 7-13):

- **Constructs**. This relates to the precision with which a construct is expressed, such that it cannot be confused with another construct, nor can it be ambiguous.

- **Associations**. This criterion is about the relationship between Things, and the specificity, consistency, and directionality of that relationship.
- **States**. More exactly, this refers to the expected state space that the allowable states may traverse.
- **Events**. Similar to the previous point, this refers to the expected event space that is allowable in the problem domain.

The whole is more than the sum of the parts, and displays emergent qualities that may not be seen in the parts. It can be evaluated according to the following criteria (2012, p. 13-16):

- **Importance**. This criterion, which is largely subjective, relates to whether the theory is important to practice or research — or, possibly, whether it provides *"deep insights"* (2012, p. 14) into phenomena.
- **Novelty**. This is about whether the theory is, in some way, transformative in the field.
- **Parsimony**. A theory that uses a small number of constructs and associations, yet achieves reasonable explanatory power, is superior to one which uses a large number of constructs and associations.
- **Level**. A very narrowly-focused, constrained theory is termed *micro-level*; a very broad, general theory is called *macro-level*. Between these two levels is the *meso-level*.
- **Falsifiability**. This is about whether the theory makes it easy to find counter-examples which would falsify it or, conversely, increase the support given to it.

3 ONTOLOGY

This section proceeds in three parts. Firstly, the notions of *types* and the *functional paradigm* is briefly introduced. This notion provides the theoretical underpinning for the ontology, and the referenced literature will be useful for readers who are interested in extending the ontology in the future. The ontology itself is then described, along with a notation that permits concise representation.

3.1 Types

A *type* associates a set of operations with a particular category, where each category has a particular set of valid values that it can take on. For example, an integer is a category that can have a range of valid values, such as 1337 or 42, and which has a set of operations (such as addition or subtraction) which are defined over it. A different kind of category, such as *"boolean"*, is associated with a different set of valid values and a different set of operations. It is the job of a *type system* to ensure that the different rules governing type systems are obeyed such that it is illegal to perform a nonsensical operation such as *"4 + true"* or *"false/"hello""*.

Weber's ontology, as described in Section 2.1, would be recognized by any computer science practitioner as a variant of an object-oriented type system. Object-orientation, in turn, relies on objects and their interactions—and it is type systems that permit objects to be modeled appropriately. To rephrase Weber's ontology in information systems terms, a problem domain is modeled by objects with attributes, specialized from classes and mutable over time via methods that determine the validity of the messages that are received, thus mediating access to their internal state. The only reason that such a short rephrasing is possible is because of the common language of types that computer scientists know.

Type systems have been in development for decades and it is impossible to cover the wealth of literature or the formal algebra of types in the space permitted. Fortunately, a formal treatment of the subject is not necessary for an understanding of the proposed ontology. This work will therefore deal with types on an informal basis and specify types only in terms of allowable operations and valid values, and in the same style as that used by Weber (2012). Interested readers are directed to Cardelli's eminently readable introduction (2004) or Mitchell's more in-depth treatment of the same (1996), and to the respective seminal works of Milner (1978) and Dijkstra *et al.* (1987). Type systems are practically implemented in many languages and have typically been linked to type-checking algorithms, and their expressive power has been curtailed by the efficiency of such algorithms. This restriction is largely irrelevant when defining an ontology, and that fact can be used to be more free with typing than might otherwise be possible.

3.2 The functional paradigm

Lisp, the first functional language, was developed in 1958 and was based on the pioneering theoretical work of Alonzo Church (Hindley & Seldin, 2008). Lisp's conception of first-class functions—values which accepted a value and generated another value—was starkly different to the way in which computation was understood in other popular languages of the time such as FORTRAN, ALGOL, and COBOL. These latter languages were imperative in nature: instead of declaring *what* was to be done in terms of operations or relationships, a programmer would define *how* computation would proceed as steps to be performed on data. The hardware of the time was much more suited to the imperative paradigm than it was to the functional paradigm, and Lisp did not gain much traction outside of niche markets.

Lisp was, however, influential in that it placed the focus on functions and simply treated them as special kinds of values. Functions could be used in the same contexts that any other kinds of values could be used; functions could be named or unnamed; functions could be defined in one place, passed through a program, and used in an entirely different place. No objects were necessary: functions took on all of the work. This reliance on functions came to be intimately associated with the functional paradigm.

Lisp had no strong conception of types and it was only in the early 1980's that a typed version of lambda calculus was developed (Milner, 1978). This led to a resurgence of interest in functional languages and also to a great deal of research into types. Most importantly from the perspective of this work, functions were given types that reflected their input and output, and it was discovered that this practise made it somewhat easier to design systems within the functional paradigm. The typing of functions did not change what they were and they still retained all of their existing characteristics as described above. Haskell is an example of a relatively mainstream functional language which lies at the intersection of these research areas (Jones, 2003).

The functional paradigm entails two things: immutable values (i.e. values which cannot be changed after they are created) and functions (as described above). Progress is made by transforming values through the application of functions. Functional programming today is becoming more and more mainstream with languages such as ECMAScript 6 (Zakas, 2016), C# (Troelsen, 2008), and Java (Warburton, 2014) all beginning to support functional paradigm style functions.

3.3 Proposed ontology

The proposed ontology does not use the *"object"* paradigm, but instead takes its inspiration from the typed lambda calculus (Milner, 1978) and the concepts of functional programming. Unlike in object orientation, the ontology does not begin by attempting to model the world. Rather, it begins by specifying primitives which, when combined, can flexibly and efficiently describe entities in the world. These entities are then combined and composed to represent a system of relations. The notation below is made up, though it is influenced by various functional programming languages.

1. **Type**. A type may fall into one of seven categories:

   (a) A *concrete* type, such as "integer" or "float". These may be used to model simple values, such as the age of participants.

   (b) A *function* type, such as "integer $\mapsto$ boolean". This type is the workhorse of the ontology: when given the type on the left of the "$\mapsto$", it gives back the type on the right of the "$\mapsto$". The left side is called the function *input*, and the right is the function *output*. The function type is used to model processes.

   (c) A *tuple* type, such as "(integer, float, string)". This is used to group together types into an ordered set. Types are separated by ",". A tuple type is typically used to model data which has no separate semantic identity in the researcher's estimation, but which happens to be grouped together at certain junctures.

   (d) A *record* type, such as "{ age:integer; risk:integer $\mapsto$ float }". This is used to group together types into a named, unordered set. Types are separated by a ";". A record type $\mathscr{X}$ that contains a superset of the fields of a record type $\mathscr{Y}$ can be used wherever $\mathscr{Y}$ is expected. It is used to model semantic groupings of data, such as the medical risk profile of someone based on their age.

   (e) A *variant* type, such as "Credit float | Debit float | Rejected". This type defines a total, discrete set of cases, where each case can be linked to another type. The variant type can be used to identify the legitimate inputs to a process or outputs of a process. Where the set is expected to be extended within a particular context, the special case "..." may be used at the end. Where modification of the stated cases is allowed as well as extension, "..." should be placed at the start.

(f) A *list* type, such as "[{ role:string; skilled:boolean }]". This is used to represent a collection of the named type.

(g) A *parametric* type, such as "$\underline{a}$". A parametric type can be substituted by any other type, but the same parametric name in the context of a function type must always represent the same type.

2. **Names**. A type may, for convenience, be aliased by a name through the use of notation such as "type count = [boolean] $\mapsto$ int". The name can always be replaced by the full type that it is linked to without altering the semantics of a theory. A name typically expresses the purpose of the type.

The expressive power of the ontology is exhibited in the way these types can be combined: any kind of type may be used in any context where a type is expected. This means, for example, that a function output may itself be a function, or that a variant may be linked to a record or another variant or even a parametric type. The only constraint on such combination is that it must be correctly typed. For example, a boolean cannot be passed as input to a function which takes an integer as input. Where ambiguity of notation might exist, or where additional explicitness is desired, ordinary brackets may be used to clarify the notation.

Function types represent processes and, as such, are critical to the ontology. They may be combined using the *composition* operator to create a new function which is the mathematical composition of the given functions. For example, given a function "type f = boolean $\mapsto$ integer" and a function "type g = integer $\mapsto$ float", it is possible to create a "boolean $\mapsto$ float" function using the composition of "f" and "g". Composition is denoted by the symbol $\gg$, so the composition of "f" and "g" would be written as "f $\gg$ g"[2].

4 CASE STUDIES

An ontology is only as good as its practical ability to represent a theory. With this in mind, as examples of the ontology's suitability, two theories are formulated in terms of the above ontology and notation: Ajzen's theory of planned behaviour as expressed in (Ajzen, 2005), and the Capability Approach is expressed in Sen, 1993. These theories have been selected on the bases of being non-trivial, well-specified, and of relevance to the Information Systems community at large (see, for example, Barth and de Jong (2017), Chang and Chen (2014), Clark (2005), Dinev and Hart (2006), Posey, Lowry, Roberts, and Ellis (2010), Zheng (2009)). As the text of a theory is encountered, the model will be refined. This refinement process demonstrates the ease with which the ontology can accommodate change. A discussion section follows after the case studies.

[2]In mathematics, $(f \circ g)(x)$ means $f(g(x))$. However, $(f \gg g)(x)$ means $g(f(x))$. While the same essential meaning present in this ontology, the ordering of function application differs, and a different symbol has therefore been used.

Both of the case studies, quite deliberately, are *not* concerned with the development of computer systems. Typed functional programming has been used to create computer systems since the introduction of Standard ML in 1983 and may arguably even predate the early 80's (Gordon, 2000). The application of functional programming to computational problems is well-tilled ground, and there is little value in demonstrating what has already been exhaustively demonstrated. For example, there are many excellent books on how to create computer systems using functional programming today (see, for example, Wlaschin (2018)).

4.1 Ajzen's Theory of Planned Behaviour

From Ajzen (2005, p. 117-118):

> According to the theory of planned behavior, intentions (and behaviors) are a function of three basic determinants, one personal in nature, one reflecting social influence, and a third dealing with issues of control. The personal factor is the individual's *attitude towards the behavior* [...]. The second determinant of intention is the person's perception of social pressure to perform or not perform the behavior under consideration. [...] Finally, the third determinant of intentions is the sense of self-efficacy or ability to perform the behavior of interest, termed *perceived behavioural control*.

Since there are three factors which work together to determine whether an intention will exist, the factors should be grouped into a single type: "type factors = (Attitude integer, Norm integer, Perceived_Control integer)", where a positive/negative integer indicates whether each factor has a positive or negative impact. Single-case variants are useful in this case because it attaches a semantic meaning to each integer, and prevents it from being used in any context where an integer—and not an attitude, norm, or perceived behavioural control—is desired. The core of the theory can then be represented as a function: "type has_intention = factors ↦ Intention boolean". Note that "Intention boolean" is a single-case variant type which, if desired, could have been given a separate name.

From Ajzen (2005, p. 118):

> For some intentions attitudinal considerations are more important than normative considerations, while for other intentions normative considerations predominate. Similarly, [...] perceived behavioral control is more important for some behaviors than for others. In some instances, only one or two of the factors are needed to explain the intention, while in others, all three factors are important determinants. In addition, the relative weights of the three factors may vary from one person to another, or from one population to another.

The above quotation explains how the factors arise: they depend on the behavioural intention, person, and population. The model can be extended by creating a type to accept an intention, a person, and a population, and produce a "factors" type. Of course, to do this, types to represent intentions, people, and populations must be created.

An intention could be represented as a variant: "type intention = ... | Eat | Drink | MakeMerry", where the actual cases are significant for the behaviours that are being studied. Each person, being unique, can be represented by a unique integer within a single-case variant: "type person = Person integer". Lastly, a population might be represented as a variant: "type population = ... | Irish | English | Argentinian". The factors-generating function can now be created: "type weighting = (intention, person, population) ↦ factors".

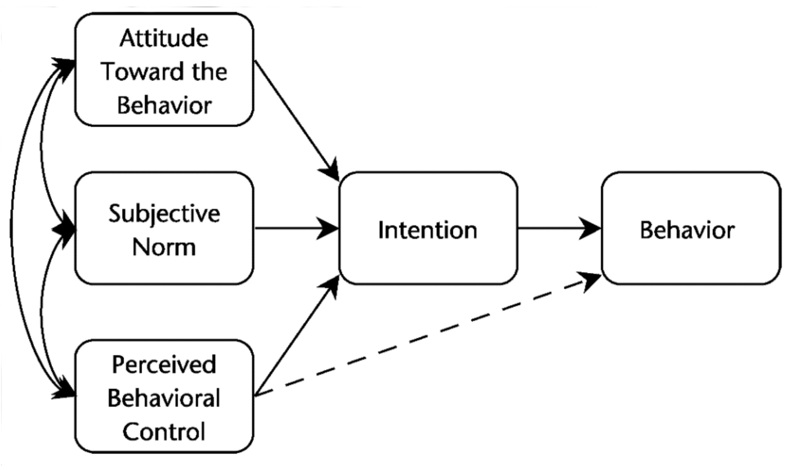

Figure 1: The Theory of Planned Behaviour (reproduced from Ajzen (2005, p. 118))

Ajzen uses a diagram (reproduced as Figure 1 here) to point out features of the theory. The most interesting part, as far as this work is concerned, is that the intention is explicitly linked to a behaviour. However, this is mediated by the *actual* ability of the person to perform the action, and the action itself may therefore not occur despite the intention being present. None of this is represented in the model that has thus far been developed. To represent this, the link between intention and behaviour can be represented as "type has_behaviour = (Actual_Ability integer, Intention boolean) ↦ Behaviour boolean".

This concludes the modelling of the theory within the proposed ontology. In summary, the following seven notations, coupled with annotations that describe the semantics of each name, fully express the theory:

1. type factors = (Attitude integer, Norm integer, Perceived_Ability integer)
2. type intention = ... | Eat | Drink | MakeMerry
3. type person = Person integer
4. type population = ... | Irish | English | Argentinian
5. type weighting = (intention, person, population) ↦ factors
6. type has_intention = factors ↦ Intention boolean

7. type has_behaviour = (Actual_Ability integer, Intention boolean) ↦ Behaviour boolean

4.2 Sen's Capability Approach

From Sen (1993, p. 31):

> *Functionings* represent parts of the state of a person—in particular the various things that he or she manages to do or be in leading a life.

A functioning can be represented within a bounded and situated context as a variant, and we can conveniently give it a name: "type functioning = ... | GoodHealth | Nourished | SelfActualised | SociallyIntegrated". Recall that the special case "...", placed at the start of the notation, indicates that a particular context may have more or less or different functionings, at the discretion of a researcher; and each case of the variant may be associated with further data. For example, the "Nourished" case can be replaced by "Nourishment float" to express the degree of nourishment, if a researcher is interested in that level of detail.

From Sen (1993, p. 31):

> The *capability* of a person reflects the alternative combinations of functionings the person can achieve, and from which he or she can choose one collection.

Assume that each person is denoted by a different integer, and provided with its own single-case variant: "type person = Person integer". Then the alternative capability sets of a particular person may be expressed the function "type capability = person ↦ [[functioning]]". Note that this is a list of lists, which reflects the idea of alternative combinations which *might* be achieved. The creation of a single-case variant instead of simply using an integer will prevent the function from being used with an integer that can mean *anything*; instead, it must be used with an integer that specifically identifies a person.

From Sen (1993, p. 32):

> (1) *What* are the objects of value? (2) *How valuable* are the respective objects? [...] identification of the set of value-objects, with positive weights, itself precipitates a 'dominance ranking' (*x* is at least as high as *y* if it yields at least as much as *each* of the valued objects). [...] The identification of objects of value specifies what may be called an *evaluative space*.

The objects of value may be represented by a variant, similar to how a functioning was represented: "type evaluativeSpace = ... | Happiness | Fulfilment | Dignity". The determination of how valuable a particular object can be obtained via two functions: "type simpleDominance = evaluativeSpace ↦ Value integer" and "type dominance = [evaluativeSpace] ↦ Value integer". Note that the input to the former is a single case, which reflects the idea that each value-object has an independent weight, and the latter is a list, which models the idea of a *set of* value objects.

From Sen (1993, p. 35):

> We can make a fourfold classification of points of evaluative interest in assessing human advantage, based on two different distinctions. One distinction is between (1.1) the promotion of the person's *well-being*, and (1.2) the pursuit of the person's overall *agency goals*. [...] The second distinction is between (2.1) *achievement*, and (2.2) the *freedom to achieve*. The two distinctions together yield four different concepts of advantage, related to a person: (1) 'well-being achievement', (2) 'agency achievement', (3) 'well-being freedom', and (4) 'agency freedom'. [...] The assessment of each of these four types of benefit involves an evaluative exercise, but they are not the *same* evaluative exercise.

Each of these may be conveniently modeled by a different function. Advantages (1) and (2) relate to an assessment of a particular set of functionings within the evaluative space, and may both be modeled by the type "[functioning] $\mapsto$ Advantage integer" under the names "wellbeing" and "agency". Advantages (3) and (4) relate to an assessment of the freedom to choose among a particular set of functionings, and must therefore take into account the set of sets of functionings; their type could be expressed as "[[functioning]] $\mapsto$ Advantage integer" under the names "wellbeingFreedom" and "agencyFreedom".

This concludes the modeling of the theory within the proposed ontology. In summary, the following nine notations, coupled with annotations that describe the semantics of each name, fully express the theory:

1. type functioning = ... | GoodHealth | Nourished | SelfActualised | SociallyIntegrated
2. type capability = Person integer $\mapsto$ [[functioning]]
3. type evaluativeSpace = ... | Happiness | Fulfilment | Dignity
4. type simpleDominance = evaluativeSpace $\mapsto$ Value integer
5. type dominance = [evaluativeSpace] $\mapsto$ Value integer
6. type wellbeing = [functioning] $\mapsto$ Advantage integer
7. type agency = [functioning] $\mapsto$ Advantage integer
8. type wellbeingFreedom = [[functioning]] $\mapsto$ Advantage integer
9. type agencyFreedom = [[functioning]] $\mapsto$ Advantage integer

5 DISCUSSION

Have the theories in the case studies been modeled correctly and completely? The authors believe that they have been, but whether they have been or not is beside the point. The real point is that the above notations present unambiguous interpretations of complex theories in a mere 7-9 short lines. Each line is either correct or incorrect, and a reader who has a different interpretation of the theory is able to hone in on the precise relationship(s) that are contentious. The possibility of misinterpreting the fundamentals of a theory due to poorly-chosen or poorly-understood natural language is reduced. As stated in the introduction to this work, for a theory to be useful as an abstraction in the field, there must be sufficient consensus within the field to minimise ambiguity. Such consensus is more easily built when the description of a theory does not admit ambiguity.

By way of comparison, consider Figure 1, which is a pictorial representation of the Theory of Planned Behaviour. Although the text indicates that behavioural intention, person, and population affect the factors, this is not represented on the diagram; and although the dotted line traces from *perceived* behavioural control, it should properly represent *actual* behavioural control. Neither of these is clear. Even more confusingly, the arrows that link attitudes, norms, and perceived behavioural control mean “is weighted relative to”, and the arrows that go towards intention and behaviour mean “influences”. The issue here is not that the diagram is poor; indeed, it seems fair to say that the diagram does as good a job as it can. The issue is that the diagram represents a set of constructs, but the constructs are often not what is important about a theory. The important part of a theory is often the relationships between entities and this can best be represented by an explicit notation.

The notation also captures, very succinctly, properties of the theory that are difficult to express in English. For example, consider this passage about the Capability Approach (Sen, 1993, p. 38):

> We should first note that capabilities are defined derivatively from functionings. In the space of functionings any point, representing an *n*-tuple of functionings, reflects a combination of the person’s doings and beings, relevant to the exercise. The capability is a *set* of such functioning *n*-tuples, representing the various alternative combinations of beings and doings any one (combination) of which the person can choose. Capability is thus defined in the *space* of functionings.

The English text is neither easy to read nor understand, yet it is expressed without ambiguity via the name “capability” in the provided modeling.

This model of the theory also exposes gaps in the relations between types. What, for example, is the exact relationship between “evaluativeSpace” and “capability”? Why is there no role for “dominance” in any of the last four functions? How is “dominance” related to “capability”? These are questions that the theory does not provide answers to. They are easily exposed through the rigorous notation of the proposed ontology.

The proposed ontology makes it easy to extend the breadth of a theory by adding additional types. Consider, for example, the following description of a specific kind of belief in the context of Theory of Planned Behaviour (Ajzen, 2005, p. 123):

> [A]ttitude toward a behavior is determined by accessible beliefs about the consequences of the behavior, termed *behavioral beliefs*. Each behavioral belief links the behavior to a certain outcome, or to some other attribute such as the cost incurred by performing the behavior.

A behavioral belief can be defined by “type behavioural = ... | ReducedBloodPressure | Dizziness | Convenient integer”, for example. It is then possible to create a “type attitudesFromBeliefs =

behavioural ↦ Attitude integer" to represent the evaluation of beliefs to determine attitude. The breadth of the model—in other words, the breadth of behaviour that can be explained through the model—is thus extended through the addition of these types.

Extensions to the depth of a theory are also viable. Consider the "simpleDominance" function: how does it manage to take a case from the "evaluativeSpace" and map it to a "Value integer"? It could conceivably be using a utilitarian theory of value. Alternatively, it could use a consequentalist theory, or a deontological perspective, or a rights-based evaluation, or something else. *Any* of these can be used to define "simpleDominance" in depth; to put it another way, "simpleDominance" can be regarded as a composition of the relevant functions within any one of the mentioned theories.

Lastly, it is worth noting that the expressive power of the ontology has hardly been taxed at all in order to represent these theories. No record or parametric types needed to be used to represent the theories completely, nor has there been a need to use function composition.

5.1 The operational semantics of function types

A function is described entirely in terms of its input and output, and the input is used to generate the output. This means that the function type can very strongly imply a particular set of operational semantics, *independent* of the name assigned to that type. These operational semantics can be used by researchers to understand the relations of a theory much more quickly. The following list gives some of the more common types to look out for, in their most generic form, and also gives the name that is used in the functional paradigm to refer to such a type.

($\underline{a}$ ↦ boolean) ↦ [$\underline{a}$] ↦ [$\underline{a}$] is the *filter* operation. The "$\underline{a}$ ↦ boolean" function is used to determine whether to retain or discard each element of the list. A function with a similar type implies the operational semantics of reduction.

($\underline{a}$ ↦ boolean) ↦ [$\underline{a}$] ↦ $\underline{a}$ is the *find* operation. The "$\underline{a}$ ↦ boolean" function is used to obtain a single matching value from the list. A function with a similar type implies the operational semantics of selection.

($\underline{a}$ ↦ $\underline{b}$) ↦ [$\underline{a}$] ↦ [$\underline{b}$] is the *map* operation. The "$\underline{a}$ ↦ $\underline{b}$" function is used to convert each element of the [$\underline{a}$] into a corresponding $\underline{b}$. A function with a similar type implies the operational semantics of transformation or conversion, without the possibility of reduction.

[[$\underline{a}$]] ↦ [$\underline{a}$] is the *flatten* operation. It implies the reduction of dimensionality, with the possibility of retaining only distinct items.

[$\underline{a}$] ↦ $\underline{b}$ is the *fold* operation. It implies aggregation or summarization based upon some characteristic of the set of $\underline{a}$ items.

$\underline{a}$ ↦ [$\underline{b}$] is the *unfold* operation. It implies the generation of information based upon some aspect of $\underline{a}$.

The utility of these frequently-observed operations goes beyond simply making it easier for researchers to grasp relations: since each operation has a name, theories become easier to discuss using that name. It is, for example, simpler to say that the "agency" type created to model the Capability Approach *folds* functionings into advantages, or that the "capability" type *unfolds* a person into sets of functionings. These operations therefore provide a common terminology that can be useful across theories and which, being focused solely on relations, is independent of the entities themselves.

5.2 Recontextualization

The process of applying a theory to a different context is called recontextualization. In theory, a theory operates in much the same way as a statistical analysis: it produces useful results, no matter which context it is applied in, as long as it is applied correctly in that context. In practice, the correct application of a theory to a different context can be exceptionally difficult since theories refer to entities which may not exist, or which may be changed, in a different context. A different context may also contain entities that are not catered for in the original theory.

This problem is not unique to the field of information systems; for example, the difficulty that Bergene (2010) finds in the social sciences tends to also lead to real-world consequences such as those laid out by Moore and Evans (2017). The problem also disproportionately affects particular sub-fields of information systems, such as ICT4D. However, much of the difficulty has to do with the fact that *entities* have been paramount in the field's conceptualization of theory, and unless the same entity is available across contexts, recontextualization becomes difficult. Consider the following cases, where $\mathscr{A}$ denotes the original context of the theory and $\mathscr{B}$ denotes a different context:

1. If the same or similar entity does not exist between $\mathscr{A}$ and $\mathscr{B}$, but the researcher wishes to use the theory anyway, then was the entity relevant for $\mathscr{A}$ in the first place? If the theory can be applied successfully, then the answer may be "no", and it may be useful to remove it from the theory; conversely, if the research did not succeed, then it can be difficult to determine whether the problem was the theory or the context.

2. If a similar entity does exist between $\mathscr{A}$ and $\mathscr{B}$, then the entity is similar — but not the same — because some attribute is missing. Once again, one is forced to ask whether that attribute was relevant for $\mathscr{A}$ in the first place, and either a "yes" or "no" answer is undesirable.

These problems can be avoided if one is able to develop a theory that is specific to $\mathscr{B}$. That solution, however, may lead to additional problems for the field as a whole since researchers who had n theories to consider will now have $n+1$ theories instead!

The proposed ontology does not solve the problem entirely, but may reduce the issues involved. This, once again, is due to the proposed ontology's emphasis on functions rather than entities.

In the trivial case, one is able to find the same processes in the new context that one was able to find in the old context, and thus use functions with the same types. Application of the theory is then a simple matter of altering type names.

In the more complex case, one or more processes may not exist in the new context. In such a case, the solution is to create a recontextualizing theory which maps the inputs of $\mathscr{B}$ to the domain of $\mathscr{A}$, and the outputs of $\mathscr{A}$ to the domain of $\mathscr{B}$. As the research proceeds, the original theory remains entirely unchanged, but the recontextualizing theory may be altered. The benefit of this is that the context and the recontextualization of the theory to suit the context are now explicitly separated. If the research succeeds, then no change needs to be made to the theory, and other research can simply use the same recontextualizing theory whenever the same context is encountered. This, in turn, makes it easier to build upon existing research rather than striking out on one's own. If the research fails, then one has the option of modifying the recontextualizing theory until success is encountered. If success is nevertheless elusive, then the original theory is increasingly falsified. This makes it much easier to determine when it is more likely that the context cannot be recontextualized for the theory, and when it is more likely that the original theory requires modification.

6 EVALUATION

It is now necessary to evaluate the proposed ontology in terms of Weber's framework (see Section 2.1) and compare it to the Bunge-Wand-Weber ontology. Given that the "parts" of a theory "circumscribe the *boundary* or *domain* of the theory"(Weber, 2012, p. 6), it is reasonable to say that these would be the different types in the proposed ontology. The "whole" of the theory is the different constructs, the fixed rules that describe how they may be legitimately combined, and the composition operator.

The proposed ontology meets the Constructs criterion to a higher degree than the Bunge-Wand-Weber ontology. Each type is precisely specified, and the ways in which types may be composed are clear; no type can be mistaken for another. There is greater scope for confusion in the Bunge-Wand-Weber ontology. For example, consider the way in which *class* and *mutual attributes* are defined (2012, p. 3):

> ***Class***. *Things that possess at least one property in common constitute a class of things. For example, all humans who use an information system are members of the class of things called "information system users."*
>
> [...]
>
> *Mutual attributes* represent properties of two or more particular things (mutual attribute in particular) or two or more classes of things (mutual attribute in general). For example, system analysts and information system users have the mutual attribute in general called "level of shared understanding about the requirements for a new information system," which will take on a specific value (mutual attribute in particular) for a specific system analyst-information system user pair).

Why does "mutual attribute in general", which specifies a property common to more than one class, not form a class of its own, by the definition of "Class"? Difficulties also arise when considering the difference between "Composite Thing"/"Property" and "Thing"/"complex attribute".

The Associations criterion is also better met by the proposed ontology. Associations are modelled by functions, each of which has an input and an output; directionality is always present, the inputs and outputs are specific types, and an association either exists or does not exist. By contrast, there is no construct that directly models associations in the Bunge-Wand-Weber ontology.

Similarly, the Bunge-Wand-Weber ontology does not provide any construct that specifically models a state space. While the constructs of “State”, “Lawful State”, “Event”, and “Lawful Event” do exist, they are represented purely as exhaustive vectors rather than as an explicit set of good states. The proposed ontology provides the variant type construct which defines a discrete and flexible set of named cases, thus expressing the entirety of a legitimate state space. This construct can also be used to model an event state space. Consequently, the proposed ontology meets both the States and Events criteria to a greater degree than the Bunge-Wand-Weber ontology.

This concludes the evaluation and comparison from the perspective of parts. From the perspective of the whole, the proposed ontology reflects a radically different way of conceptualising a theory. The discussion that followed the modelling of the Theory of Planned Behaviour demonstrates that there are some insights provided by such a representation that are not easily accessible when the ontology is not in use. The proposed ontology therefore meets both the Importance and Novelty criteria.

In terms of Parsimony, the proposed ontology has the upper hand. The table that lists “Some Fundamental Ontological Constructs” (Weber, 2012, p. 3-4) specifies eleven separate constructs; taking into account the composition operator, the proposed ontology specifies only three. It could be argued that each specific type is a separate construct, but if that is the case, then it would also be appropriate to regard each specific attribute as a separate construct; and under this interpretation, the Bunge-Wand-Weber ontology specifies sixteen separate construct as opposed to the proposed ontology’s nine.

The Bunge-Wand-Weber ontology is quite clearly a macro-level theory, capable of flexibly representing a number of specific theories. The same can be said of the proposed ontology. Both, therefore, are equal with regard to the Level criterion.

Falsifiability is not explicitly addressed by the Bunge-Wand-Weber ontology due to a lack of a specific construct that represents a lawful state space. Conversely, a variant which claims to exhaustively represent a particular problem domain may be tested as part of the process whose input or output it is; if it does not, in fact, fully represent the state space, then the function will be falsified.

7 CONCLUSIONS

It might be deduced from the previous section that the proposed ontology is superior to the Bunge-Wand-Weber ontology in all respects. Nothing can be further from the truth: it is more exact to say that it is *complementary* to it. There are some problem domains which are represented best by independent objects, and the Bunge-Wand-Weber ontology represents these problem domains faithfully. There are also some problem domains which are represented best by relations, and the proposed ontology may be better for representing those. In any event, having two ways to represent a particular theory can only help the Information Systems practitioner to develop more rigorous and viable theories.

Many sub-fields of Information Systems are full of objects. To regard the objects themselves as being of prime importance often leads to regarding the *specific context* of the theory to be crucial, since objects are often specific to a context. For example, Legitimation Code Theory (Maton, 2011) discusses knowledge, knowers, and disciplines in an academic context, and is bound to that context by the fact that various artifacts—such as a formal "discipline"—do not exist outside of academia. The theory may also be applicable to education within industry and to the problem of silo'ed knowledge in organizations, but it has not been applied to these contexts: its modelling through entities, rather than through relationships, makes it difficult to extract from its context.

ACKNOWLEDGEMENTS

This work is based on the research supported wholly by the National Research Foundation of South Africa (Grant Numbers: 116794). The helpful comments of the anonymous reviewers are gratefully acknowledged. A previous version of this research was presented at SAICSIT 2018 (Motara & van der Schyff, 2018).

References

Ajzen, I. (2005). *Attitudes, personality and behavior* (I. Ajzen, Ed.). New York, United States of America: McGraw-Hill Education.

Barth, S., & de Jong, M. D. (2017). The privacy paradox: Investigating discrepancies between expressed privacy concerns and actual online behavior – A systematic literature review. *Telematics and informatics*, *34*(7), 1038–1058.

Bergene, A. (2010). Towards a critical realist comparative methodology. *Journal of critical realism*, *6*(1), 5–27.

Bunge, M. (1977). *Treatise on basic philosophy: Ontology I: The furniture of the world* (M. Bunge, Ed.). Berlin, Germany: Springer.

Cardelli, L. (2004). Type systems. In *Handbook of computer science and engineering*. Digital Equipment Corporation.

Castañeda, H.-N. (1972). Plato's "Phaedo" theory of relations. *Journal of philosophical logic*, *1*(3/4), 467–480.

Chang, C. W., & Chen, G. M. (2014). College students' disclosure of location-related information on Facebook. *Computers in human behavior*, *35*, 33–38.

Clark, D. A. (2005). Sen's capability approach and the many spaces of human well-being. *Journal of Development Studies*, *41*(8), 1339–1368.

Dahl, O. J., Dijkstra, E. W., & Hoare, C. A. R. (1987). *Structured programming* (C. A. R. Hoare, Ed.). New York, United States of America: Academic Press.

Dinev, T., & Hart, P. (2006). An extended privacy calculus model for e-commerce transactions. *Information systems research*, *17*(1), 61–80.

Fine, G. (2003). *Plato on knowledge and forms: Selected essays*. Oxford, United Kingdom: Oxford University Press.

Gašević, D., Guizzardi, G., Taveter, K., & Wagner, G. (2010). Vocabularies, ontologies, and rules for enterprise and business process modeling and management. *Information systems*, *35*(4), 375–378.

Gordon, M. (2000). Proof, language, and interaction. In E. Milner (Ed.), *From LCF to HOL: A short history*. Cambridge, United States of America: MIT Press.

Gregor, S. (2006). The nature of theory in information systems. *MIS quarterly*, *30*(3), 611–642.

Heffernan, C., Lin, Y., & Thomson, K. (2016). Drawing from development: Towards unifying theory and practice of ICT4D. *Journal of international development*, *28*(6), 902–918.

Hindley, J. R., & Seldin, J. P. (2008). *Lambda-calculus and combinators, an introduction*. Cambridge, United Kingdom: Cambridge University Press.

Jones, S. P. (2003). *Haskell 98 language and libraries—The revised report* (S. Jones, Ed.). Cambridge, United Kingdom: Cambridge University Press.

Maton, K. (2011). Theories and things. In *Disciplinarity: Functional linguistic and sociological perspectives*.

Milner, R. (1978). A theory of type polymorphism in programming. *Journal of computer and system sciences*, *17*(3), 348–375.

Mitchell, J. C. (1996). *Foundations for programming languages* (J. Mitchell, Ed.). Cambridge, United States of America: MIT Press.

Moore, G. F., & Evans, R. E. (2017). What theory, for whom and in which context? Reflections on the application of theory in the development and evaluation of complex population health interventions. *SSM - Population health*, *3*, 132–135.

Motara, Y. M., & van der Schyff, K. (2018). A functional ontology. In *Proceedings of the 2018 Conference of the South African Institute of Computer Scientists and Information Technologists* (pp. 49–54). New York, NY, USA.

Neumann, R., Parry, S., & Becher, T. (2002). Teaching and learning in their disciplinary contexts: A conceptual analysis. *Studies in higher education*, *27*(4), 405–417.

Posey, C., Lowry, P. B., Roberts, T. L., & Ellis, T. S. (2010). Proposing the online community self-disclosure model: The case of working professionals in France and the U.K. who use online communities. *European journal of information systems*, *19*(2), 181–195.

Sen, A. (1993). Capability and well-being. In M. Nussbaum & A. Sen (Eds.), *The Quality of Life*. Oxford, United Kingdom: Clarendon Press.

Troelsen, A. (2008). *Pro C# 2008 and the .NET 3.5 platform* (E. Buckingham, Ed.). New York, United States of America: Apress.

Wand, Y., & Weber, R. (1993). On the ontological expressiveness of information systems analysis and design grammars. *Information systems journal*, *3*(4), 217–237.

Warburton, R. (2014). *Java 8 lambdas* (M. Blanchette, Ed.). Los Angeles, United States of America: O'Reilly.

Weber, R. (2012). Evaluating and developing theories in the information systems discipline. *Journal of the Association for Information Systems*, *13*(1), 1–30.

Weber, R. (2016). Thirty years of the Journal of Information Systems: Reflections of a prodigal son. *Journal of information systems*, *30*(1), 137–146.

Weick, K. E. (1995). What theory is not, theorizing is. *Administrative science quarterly*, *40*(3), 385–390.

Wlaschin, S. (2018). *Domain modeling made functional*. Berlin, Germany: The Pragmatic Bookshelf.

Zakas, N. C. (2016). *Understanding ECMA Script 6: The definitive guide for JavaScript developers*. New York, United States of America: No Starch Press.

Zheng, Y. (2009). Different spaces for e-development: What can we learn from the capability approach? *Information technology for development*, *15*(2), 66–82.

zur Muehlen, M., & Indulska, M. (2010). Modeling languages for business processes and business rules: A representational analysis. *Information systems*, *35*(4), 379–390.