# Ontology then Agentology: A Finer Grained Framework for Enterprise Modelling

Chris Partridge[1,2], Sergio de Cesare[1], Andrew Mitchell[2], Ana León[4], Frederik Gailly[3] and Mesbah Khan[5]
[1]*University of Westminster, U.K.*
[2]*BORO Solutions Ltd., U.K.*
[3]*Faculty of Economics and Business Administration, Ghent University, Belgium*
[4]*Universitat Politècnica de València, Spain*
[5]*Tullow Oil plc., U.K.*

Keywords: Enterprise Modelling, Computation Independent Model, Ontology, Agentology, Essential Indexical, De Se, De Re, De Se Knowledge, De Re Knowledge, Deitic Centre, Agent-Neutral, Agent-Relative.

Abstract: Data integration of enterprise systems typically involves combining heterogeneous data residing in different sources into a unified, homogeneous whole. This heterogeneity takes many forms and there are all sorts of significant practical and theoretical challenges to managing this, particularly at the semantic level. In this paper, we consider a type of semantic heterogeneity that is common in Model Driven Architecture (MDA) Computation Independent Models (CIM); one that arises due to the data's dependence upon the system it resides in. There seems to be no relevant work on this topic in Conceptual Modelling, so we draw upon research done in philosophy and linguistics on formalizing pure indexicals – 'I', 'here' and 'now' – also known as de se (Latin 'of oneself') or the deitic centre. This reveals firstly that the core dependency is essential when the system is agentive and the rest of the dependency can be designed away. In the context of MDA, this suggests a natural architectural layering; where a new concern 'system dependence' is introduced and used to divide the CIM model into two parts; a system independent ontology model and a system dependent agentology model. We also show how this dependence complicates the integration process – but, interestingly, not reuse in the same context. We explain how this complication usually provides good pragmatic reasons for maximizing the ontology content in an 'Ontology First', or 'Ontology then Agentology' approach.

## 1 INTRODUCTION

Data integration of enterprise systems typically involves combining heterogeneous data residing in different sources into a unified, homogeneous whole. This heterogeneity takes many forms and there are many significant practical and theoretical challenges to managing it, particularly at the semantic level. In this paper, we consider a type of semantic heterogeneity that is common in Model Driven Architecture (MDA) Computation Independent Models (CIM); one that arises due to the data's dependence upon the system it resides in. There seems to be no relevant work on this topic in the Conceptual Modelling literature, so we draw upon research done in philosophy and linguistics on formalizing pure indexicals – 'I', 'here' and 'now' – also known as *de se* (Latin 'of oneself') or the *deitic centre*. This reveals firstly that the core dependency is essential when the system is agentive and that the rest of the dependency can be designed away. In the context of MDA, this suggests a natural architectural layering; where a new concern 'system dependence' is introduced and used to divide the CIM model into two parts; a system independent ontology model and a system dependent agentology model. We also show how this dependence complicates the integration process – but, interestingly, not reuse in the same context. We explain how this complication usually motivates maximizing the (domain) ontology content in an 'Ontology First', or 'Ontology then Agentology' approach.



Partridge, C., Cesare, S., Mitchell, A., León, A., Gailly, F. and Khan, M.
Ontology then Agentology: A Finer Grained Framework for Enterprise Modelling.
DOI: 10.5220/0006606304540463
In *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2018)*, pages 454-463
ISBN: 978-989-758-283-7

In this introduction, we give an overall context for the paper. Then we establish the broad features of system dependence by reviewing the work on de se done in philosophy and linguistics – where its subjects are human. Then we show how this applies in the related case of enterprise systems, using an extended simple example. With the argument established, we then look at its methodological and architectural implications. Finally, we make some brief comments on future work and summarize the paper.

## 1.1 MDA Architectural Layers

The Object Management Group (OMG) has produced significant documentation of the mainstream approach to MDA. This starts with a general notion of a system (here we narrow our focus to enterprise application software systems). Then, they relate this to models, where, for them, "[a] model in the context of MDA is information selectively representing some aspect of a system based on a specific set of concerns. The model is related to the system by an explicit or implicit mapping." (Object Management Group, 2003) an almost identical statement is in (ORMSC, ORMSC Draft). Interestingly for us, this text recognises that there is a relation from the model to the modelled system and that it is not always explicit; but it does not mention the relation from the system's data to the system.

OMG then outlines the need for the models to work in architectural layers based upon separating sets of concerns; recognising that: "Separation of concerns enables greater agility, ability to deal with change and a "divide and conquer" approach to realizing a system." (Object Management Group, 2014). They accept that "there can be any number of architectural layers" and identify these three possible layers in Table 1. As noted earlier, we focus on the first of these three, the CIM.

## 1.2 Data Integration Configuration

The integration of data in enterprise systems typically involves combining heterogeneous data residing in different sources into a unified whole; where a significant part of the process is the transformation of the data into a homogeneous format. There are a variety of possible integration configurations. Data warehouses extract data from a variety of source systems, transform it into a common, homogeneous format and load this into a target warehouse.

## 1.3 Direct Interoperability Configuration

Here we are focussed on the CIM level and so use in our examples a type of integration scenario that allows the CIM to vary while keeping the other levels (PIM and PSM) unchanged. These scenarios involve the integration of data across a group of standard implementations of an application package on the same hardware. As these have the same program code and the same platform, they share a common PIM and PSM. All that can vary within the group is the system specific data content. We call such groups here *standard scenarios*.

Data that can be copied directly from one system to another within a standard scenario without causing an operational error is called directly interoperable. We use examples to illustrate the way dependence works by showing system independent data is directly interoperable while the system dependent data is not. This is a kind of extension of Leibniz's intersubstitutivity *salva veritate* principle – so we could say that dependent data is interoperably opaque.

## 1.4 System Dependent Data

It is relatively easy to find examples of system dependent data; data that is dependent upon the system of which it is a part. Take a standard scenario and consider one of the packages. Assume it has a system configuration file (this is likely to contain system-dependent data). Assume further that this configuration has a 'System Base Currency' attribute with a value of 'USD'. This 'means' that the base currency for that specific system is US Dollars.

Table 1: OMG's MDA architectural layers (adapted from (Object Management Group, 2014)).

| Name | Acronym | Description |
|---|---|---|
| **Computation Independent Model** | CIM | Business or domain models – models of the actual people, places, things, and laws of a domain. |
| **Platform Independent Model** | PIM | Logical system models – models of the way the components of a system interact with each other – independently of the platform upon which they are implemented. |

| | | |
|---|---|---|
| **Platform Specific Model** | PSM | Implementation models for a specific type of platform; for the set of resources on which a system is realized. |

One can get a feel for its dependence by considering a couple of integration scenarios where there is no transformation. If this data is copied to another directly interoperable system without transformation, clearly there is no guarantee that it will be correct as the new system may well have a different base currency. Consider those packages in the standard scenario that have the same 'USD' value in the equivalent attribute. At a data level, their content is equivalent, there are no differences. However, one cannot integrate these different systems into a single target without transformation as, although the data looks exactly the same, it does not mean exactly the same. In each case, the data is saying that US Dollars is the base currency of that specific system; in other words, it is dependent upon its owning system.

## 1.5 System Independent Data

It is equally easy to find examples of system independent data. Consider the application package above, and assume it has a currency table with a 'USD' row. Then the equivalent currency tables in some other implementations are likely to have a similar 'USD' row. In normal circumstances, these two rows are homogenous (in the data integration sense), they 'mean' the same thing. So it is likely that one could safely simply copy the 'USD' row into the equivalent table of other package systems without causing problems.

## 1.6 The Dependence Distinction in Software Engineering

Most mainstream work on software engineering pays little attention to the system dependence distinction, either using language that clearly makes no commitment to the system or casually shifting from one perspective to the other. For example, Pressman (Pressman, 2005) could be taking a system independent perspective when he suggests that a model is constructed by asking the customers what are "… the "things" that the application or business process addresses".

One MDA Guide (Object Management Group, 2003) focuses on the system and its environment, saying the CIM "… describe[s] the situation in which the system will be used" and "is a model of a system that shows the system in the environment in which it will operate, and thus it helps in presenting exactly what the system is expected to do" (Section 3.1). In another (see the description in Table 1) the CIM is described in system-free terms.

There are some papers that tackle related issues; for example, a series of papers (by some of the current authors) where indexicality and the related theme of epistemology in enterprise models are explicitly discussed (Partridge, 1996), (Partridge, 2002a), (Partridge, 2002b) and (Partridge, Mitchell and De Cesare, 2012). This paper focusses explicitly on the de re – de se distinction in CIM level business.

## 1.7 Understanding System Dependence

While it is relatively easy to identify system dependent (and independent) data, we have not been able to find any research that analyses and explains this specific phenomenon in the Conceptual Modelling literature. One obvious reason is that, from the perspective of Conceptual Modelling, this system dependence could be regarded as a given, as the code is expected to run on, and so already relativized to, the system; hence there is no need to explicitly introduce it.

However, there has been extensive discussion of almost the same phenomenon in the philosophy and linguistics literature. Since Frege (Frege, 1997) and more recently, Perry (Perry, 1979) and Lewis (Lewis, 1979) (among others) there has been significant work done in philosophy and linguistics on formalizing pure indexicals – 'I', 'here' and 'now' – also known as *de se* (Latin 'of oneself') or the *deitic centre*. A commonplace of this work is that there are cases where the pure indexicals are essential, ones where they cannot be completely translated into non-indexical *de re* (Latin 'about objects') knowledge (Perry, 1979). A standard approach, in so far as there is one, to formalising these indexicals is to regard the formalization as relative to a context that includes the deitic centre.

In areas where the role of the pure indexical might be expected to be prominent – such as pervasive computing (where there is a focus on context-location) and agent computing (where there are multiple deitic centres) – a short-term pragmatic approach is taken where the use of pure indexicals is avoided by using non-indexical identifiers.

# 2 THE PHILOSOPHICAL DE SE

Philosophy and linguistics have developed a good understanding of what differentiates the pure (de se) indexical and (de re) non-indexical that we can exploit for our analysis of system dependence. A characteristic of pure (de se) indexical uses is that the reference (and truth) of a sentence can shift from use to use. For instance, if John and Mary both utter the sentence 'I am hungry', the two utterances refer to different things; that (in de re non-indexical terms) 'Mary is hungry (now)' and 'John is hungry (now)'. And there is no (logical) inconsistency in one of the utterances being true and the other false. This does not happen with non-indexical de re uses. So, for example, the reference (and truth) of the sentence 'Mary is hungry at time t' does not change whoever, wherever and whenever it is uttered - each utterance has exactly the same content.

There is a sense in which John and Mary utter the same 'I am hungry' sentence. In this paper, we will do this by saying they have the same character; broadly following the distinction between content and character in (Kaplan, 1989), where the same pure (de se) indexical utterance types have the same character, but their content (and so truth) depends upon a context – in this case who utters the sentence. Whereas (de re) non-indexical sentences always have the same content. Sentences with character (and so a context) are clearly more complicated to integrate properly than ones without.

### 2.1 The Essential (Indexical)

As the example shows, it is true that the content of pure (de se) indexical sentences can be translated into (de re) non-indexical sentences. But what Perry (Perry, 1979) and others have shown is that the translations are not complete as there is an essential core of de se knowledge that cannot be translated. A neat way of illustrating this is with a situation where someone has de re knowledge of what is happening, but has not made the link to the corresponding de se indexed knowledge. The unexpectedness of this has provided authors with a useful literary device. In Chapter 3 – 'Pooh and Piglet Go Hunting and Nearly Catch a Woozle' – of the children's book Winnie-the-Pooh (Milne, 1926), Winnie-the-Pooh follows the tracks of what he thinks might be a Woozle, until he realizes that he has been 'Foolish and Deluded' and that the tracks are his own. In this case, it is initially true for an observer to say 'Winnie-the-Pooh is following his own tracks' but not initially for Winnie-the-Pooh (himself) to say that 'Winnie-the-Pooh is following his own tracks' – as he believes that he is following someone else.

The examples are taken as clear evidence that one cannot always translate de se indexical knowledge completely into de re knowledge. As David Lewis (Lewis, 1979) puts it, the content of de re and de se knowledge is like a map and the untranslatable kernel of de se indexical knowledge is like an 'I am here' arrow marking where one is on the map. The de re map can be made as detailed as one wishes, but it still will not show the de se arrow. The map tells one about the nature of the world; the arrow tells you, in addition, where you are in that world. If one extends this analogy to multiple agents, then the potential for interoperability issues becomes clear; their de se 'I am here' knowledge obviously cannot be directly passed between them, it needs to be translated.

As the various authors (rightly) claim, the mere possibility that this can happen is sufficient to show that de re knowledge, by itself, is unable to encompass all de se knowledge. What is needed to link the two types of knowledge is what Holton (Holton, 2015) calls, *breakthrough knowledge*: a piece of knowledge that enables the two types to be connected. When Winnie-the-Pooh realizes that the tracks are his, he acquires breakthrough knowledge that enables him to connect the two bodies of knowledge and integrate them.

## 3 DE SE KNOWLEDGE IN ENTERPRISE SYSTEMS

Enterprise systems differ from people; not least in that they are artefacts. Despite the difference, there is a similar de se – de re distinction. One illustration of this is the ease with which we can recreate a similar example. Consider an enterprise system that takes as input event logs and outputs an analysis of them. Assume that, when producing the enterprise model for this system a design choice was made to exclude processing that checks whether the logs are for the system doing the processing. Now consider a situation where this system sometimes consumes its own event logs. In this case, the system is playing the same kind of role as Winnie-the-Pooh tracking the Woozle. It has a reasonably complete picture of the event logs, but does not have the breakthrough knowledge that could link some of these to itself. The problem is not one of principle. The designers of the system could just have easily designed processing that makes the link – and probably would do if there were a requirement, such as its own event log needing to trigger an action.

## 3.1 Bank Example

We now move on to illustrating how this affects enterprise models and to do this we need to look at a different, more extended, example; one that takes advantage of the artefactual nature of these systems. The example aims to illustrate the essentiality of the de se for (system) agency, and also the different nature of de re and de se data, by showing how one is directly interoperable (in the sense introduced earlier) while the other is not. In other words, we will examine whether the different types of data can be copied directly from one standard package system to another without causing an operational error.

### 3.1.1 A Naïve Neutral Modelling Notation

Our aim with the models used in this example is to show the de se and de re data embedded in enterprise systems. We want to avoid any kind of commitment to a particular style of CIM modelling to avoid any possibility that this implicitly makes some assumptions. Hence we have chosen to use a simple naïve modelling notation with minimal assumptions.

### 3.1.2 De Re View – No De Se Deitic Core

Figure 1 shows a de re view of the example. It shows three banks that we assume (for simplicity) deal in three currencies. They hold correspondent accounts with each other to facilitate the transfer of funds; only the US Dollar accounts are shown in the figure. For this example, we consider just the two transactions across these accounts shown in the figure.

In this example, we include two processes associated with correspondent accounts:

1. The administrator of the account is responsible for keeping a master record of the account transactions (and its balance) and reporting any transactions to its owner.
2. The owner of the account is responsible for keeping a copy record of the account transactions (and its balance) based upon transactions reported from its administrator.

So, for example, when, as part of the first transfer, a payment is made from Account No. 1234, its administrator, MegaBank, is responsible for recording this and advising its owner, GigaBank, so they can record this.

Nowadays, banks delegate these responsibilities to computer systems. Let us assume – as shown in Figure 1 – that the banks in our example all use instances of the same (notional) banking package, Bancology (hence they are a standard scenario and so easily illustrate direct interoperability or its lack).

This gives us enough data to recreate a similar type of issue to that found in the earlier de se examples. Assume that the Bancology system's data structures follow the de re view laid out in Figure 1.

Now consider one of the system instances – Mega-Bancology, say. The instance has no way of knowing who owns it and so what responsibilities it has been delegated. What can the system do to ascertain how to process either transaction? Given the information at hand, it cannot work out whether it has administration or owning responsibility for either leg of the transfer – or neither.

### 3.1.3 Minimal De Se Deitic Core

Only when the system is given the additional breakthrough self-ascription information – shown in Figure 2 – can it work out what to do. In this case, if the system is given the de se data that it is the

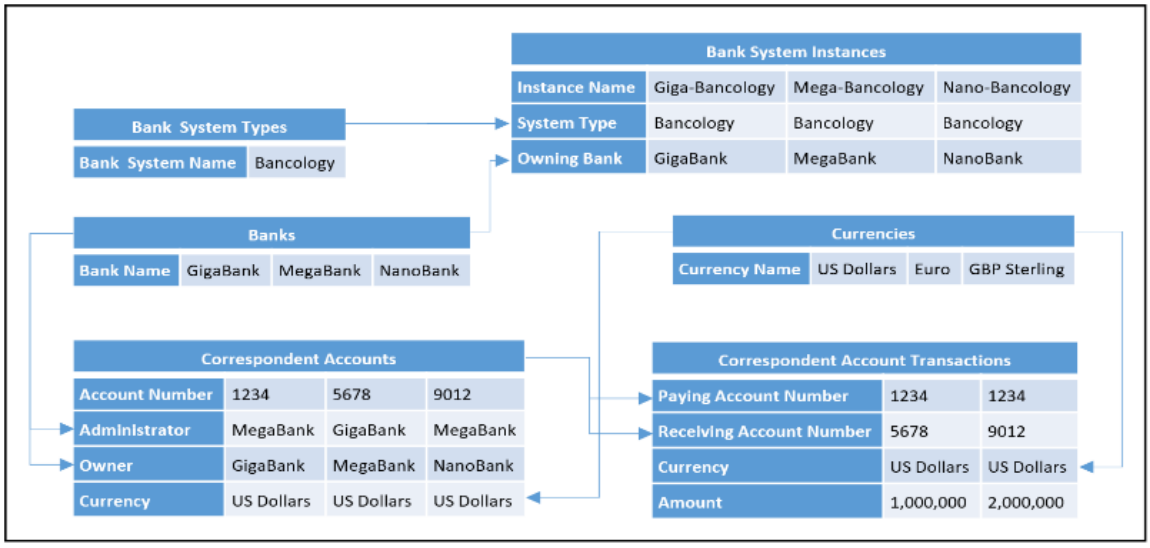


Figure 1: The de re view.

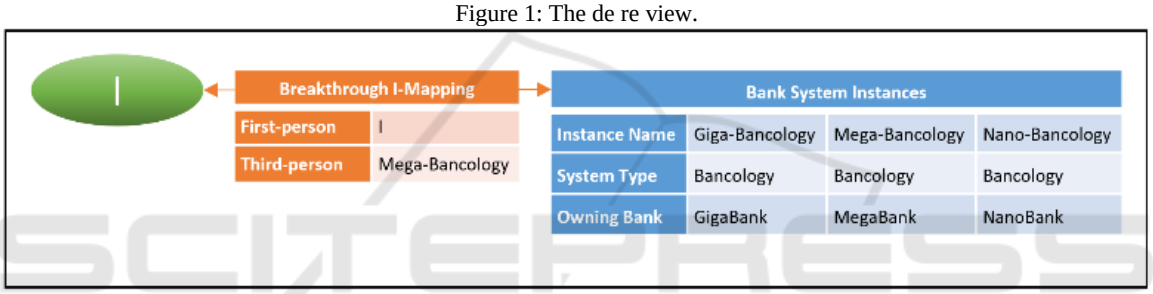


Figure 2: Breakthrough I-mapping extension.



Table 2: Perspectives on nostro-vostro and counterparty nostro.

| Account | MegaBank perspective | GigaBank perspective | NanoBank perspective |
|---|---|---|---|
| 1234 | Vostro (and GigaBank Counterparty Nostro) | Nostro | GigaBank Counterparty Nostro |
| 5678 | Nostro | Vostro (and MegaBank Counterparty Nostro) | MegaBank Counterparty Nostro |
| 9012 | Vostro (and NanoBank Counterparty Nostro) | NanoBank Counterparty Nostro | Nostro |

individual (system) Mega-Bancology, then it can infer it 'works' for MegaBank, where MegaBank is an administrator or owner, and carry out the relevant processes. The other systems would need information with the same character, but content relative to themselves. This recapitulates the essentiality of the de se for agency. As an aside, this level of self-awareness is uncommon in enterprise systems. They tend to have data structures closer to those described in the next section.

### 3.1.4 More Typical De Se View – Deitic Neighbourhood

In practice, banks tend to classify the correspondent accounts in their books relative to themselves (that is, in de se mode) as either *nostro* (Italian for 'ours') when they own the account and *vostro* (Italian for 'yours') when they administer the account. Table 2 shows this for the example's accounts. It also shows a related classification - counterparty nostro; this is the nostro account of the (trading) counterparty. As one can see, sometime this is, and sometimes is not, a correspondent account of the bank.

Let us now assume that the Bancology system has been designed to use de se nostro and vostro classifications for correspondence accounts – and also use (non-vostro) counterparty nostros: this involves introducing agent-relative types for these. This is a common design choice. We illustrate this using the Giga-Bancology instance in Figure 3. Firstly, note this has a deitic centre "I" with its link to GigaBank (this plays the same role as the earlier breakthrough I-mapping in Figure 2) – often implicit in enterprise systems or recorded on a configuration table.

There are several differences between the de se and de re views. The Banks type is agent-relative, unlike the earlier de re view, and excludes the system-owning bank. The correspondent bank accounts are divided into more specific agent-relative types. Nostro accounts are those correspondent accounts where the system-owning bank is the owner. Vostro accounts are those correspondent accounts where it is the administrator. In this agent-relative context, there is no absolute requirement for keeping a record in each 'row' of GigaBank's role – so the account owner is dropped for the nostro type and the account administrator from the vostro type – as these are always the owner of the system. Its (non-vostro) counterparty nostro accounts are the nostro accounts of its counterparties, where these are not already vostro accounts. From MegaBank's perspective, its Account No. 5678 is a nostro account. As MegaBank is a counterparty of Gigabank, this account is technically a counterparty nostro account (as shown in Table 1), however it is excluded so that the agent-relative types do not overlap.

This agent-relative perspective simplifies the processing, which can be rewritten as follows:

The system is responsible for

1. Keeping a master record of the vostro account transactions (and its balance) and reporting any transactions to its owner.
2. Keeping a copy record of the nostro account transactions (and its balance) based upon transactions reported from its administrator.

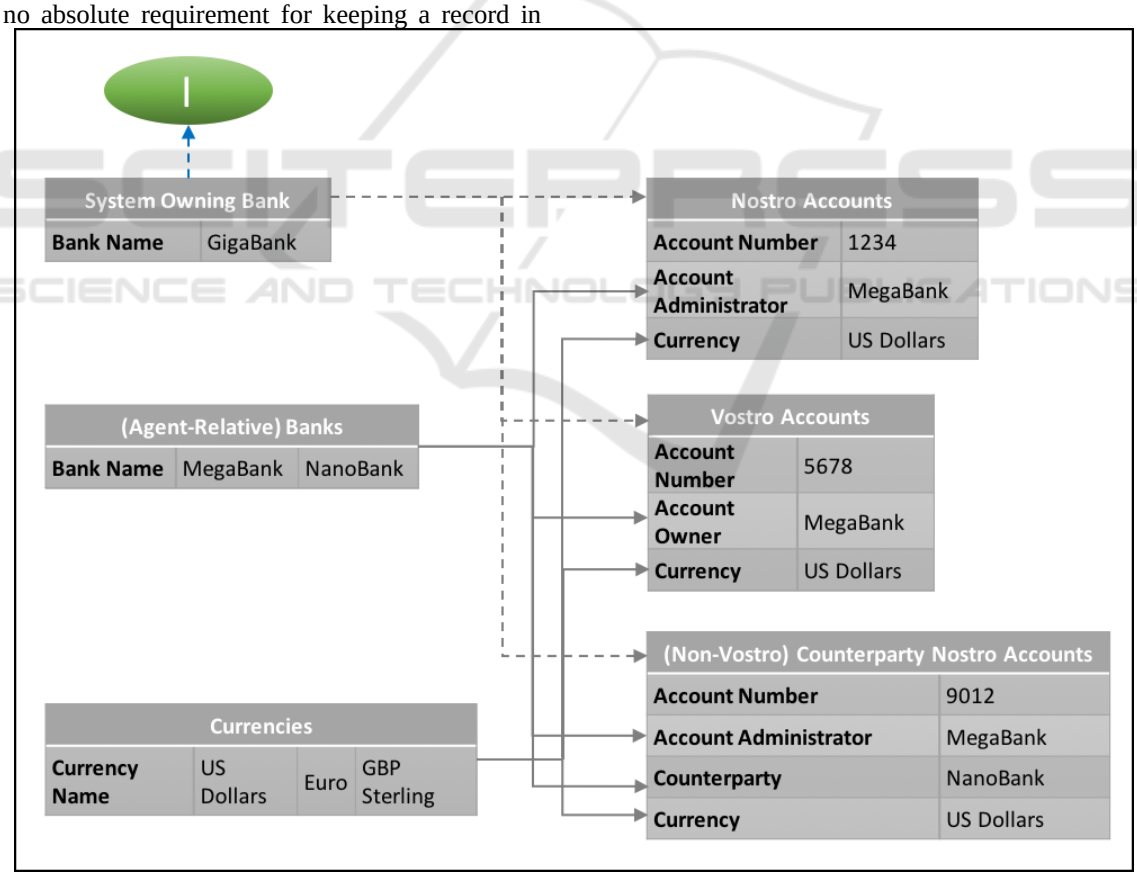


Figure 3: A more typical view - Giga-Bancology.

The agents' different responsibilities to these accounts result in different types of information in them. The vostro accounts contain the master record of the transactions and balances – so the balance is

authoritative. The nostro accounts are copies of the master records and so, while intended to be complete (include all transactions) they are less authoritative – they rely on good and timely information from the administrator. Finally, the (non-vostro) counterparty nostro accounts are not complete, as the agent will typically not know all (or even most) of the transactions – hence it makes no sense to even calculate a balance.

Now we show the Mega-Bancology instance in the same Bancology structure in Figure 4.

This, as expected, has the same character (structure) with very different content. One obvious example is the deitic centre, which links to GigaBank rather than, as in Figure 3, to MegaBank. Another interesting difference is that there are no (non-vostro) counterparty nostro accounts in Figure 4 as MegaBank's counterparty nostro accounts are all vostro accounts.

This clearly shows that the same de re data is being viewed differently by different agents. For example, there is literally no correspondent account row in common in these two agent views: none of these are classified in the same way. This illustrates how taking a fully-fledged agent-relative view (with agent-relative types) leads to each agent dividing up their view of the world (correspondent accounts) in a different way. But what is interesting here is that though the data is different, the metadata or schema is the same. This works in a similar way to language indexicals. The agent-relative view encoded in the enterprise layer of the system enables agents to build views with the same 'character' (see earlier (Kaplan, 1989) definition) but different content. In enterprise system terms, using an agent-relative view does not hinder agents with similar views from reusing the same view. Package software trades on this, enabling a standard agent-relative character to be reused by many agents. Clearly agent-neutral metadata structures can also be reused. But, maybe less obviously, an agent-relative character cannot be reused for a different character.

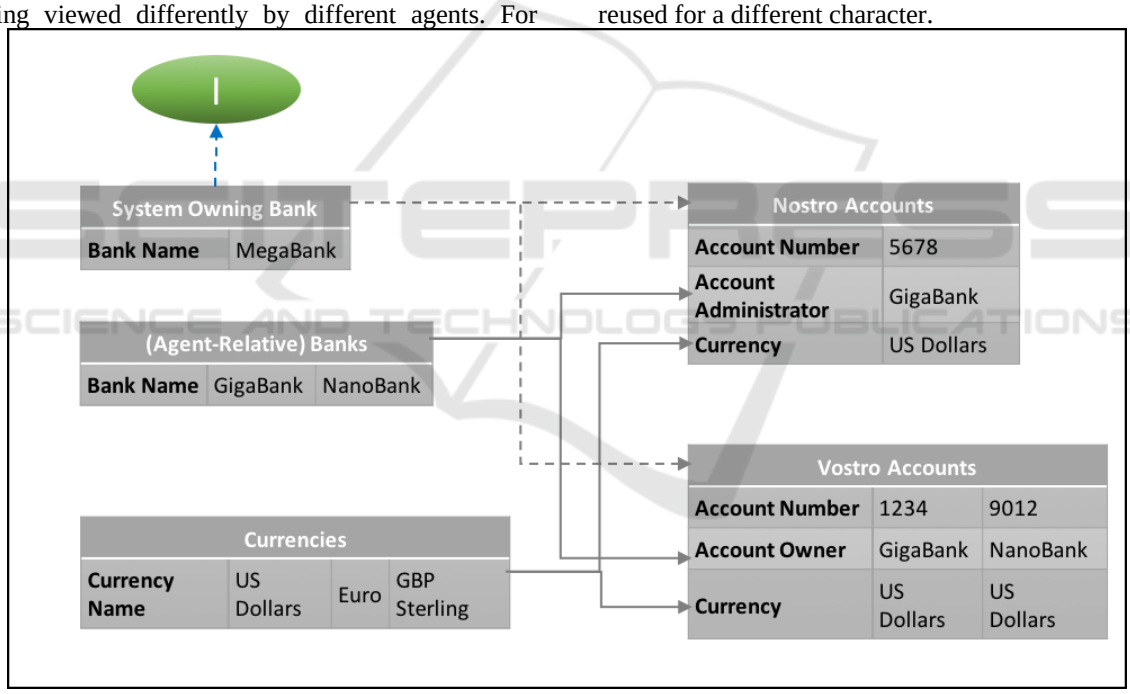


Figure 4: Another more typical view - Mega-Bancology.

## 3.2 Comparing the Two 'Models'

As already noted a few times, it is possible to translate between some – probably most – de se and de re (agent-relative and agent-neutral) data. The only intractable element is the deitic centre. The two types of model – the minimal deitic core in Figure 1 & 2 combined and the maximal deitic core in Figures 3 and 4 above – illustrate two design extremes this translation offers. The 'minimal deitic core' model has a minimal agent-neutral core - "I". The 'maximal deitic core' model is pragmatically maximal agent-relative. Later in the paper, we will revisit these design choices, when considering the enterprise (model) architecture.

## 3.3 Direct Interoperability Test for Agent-Relative Data

Reuse and interoperability are usually important design considerations in system design; semantic reuse and interoperability are usually important design considerations in enterprise (model) level system design. Here we consider direct interoperability as an instrument for distinguishing between agent-neutral and agent-relative data within systems (and so the equivalent enterprise models). This shows the different interoperability characteristics of agent-neutral and agent-relative components of the design – which, later in the paper, we use to motivate architectural design choices.

As we have already established that systems with agency have necessarily agent-relative components. These agent-relative components can be reused in a new implementation of the system, with a different agent, provided the character requirements are the same – even though they lead to different content. This reuse of the components is not compromised by these changes in content, as it works with character. Direct interoperability however is sensitive to content – so it provides a good instrument for distinguishing between agent-neutral and agent-relative data.

The direct interoperability test, in the simple cases, takes a type from two (or more) systems with the same data structures and merges their content. Obviously, in more complex cases, where there are dependencies between types, a network of types may need to be selected and then the merge may turn out to be less simple. However, we have mostly simple cases here. It then loads the merged content back into the source systems and sees whether there is any operational difference. If there is, this indicates that there is de se content.

# 4 OPEN QUESTIONS FOR AGENTOLOGY METHODOLOGY AND ARCHITECTURE

A system's enterprise concerns – and so its enterprise model – can involve representations from both de re and de se perspectives. When one starts regimenting the de re perspective a natural result is an 'ontology' – including, at least, a list of the things that exist (Partridge, 2002a), (Partridge, Mitchell and De Cesare, 2012). Given that the deitic centre is an agent, we have proposed calling its regimented perspective an '*agentology*'.

It should be clear now that where an enterprise system has agency (that is, when it can do something), it will have an irreducible deitic centre and so an underlying agentology – which can be exposed by regimentation. During the development of the system, if an enterprise model were produced then one would expect it to represent this deitic centre. As the bank example illustrates, in the deitic neighbourhood the system could have either de se or de re types. In the deitic outskirts, the types naturally lose any de se character; in the example, the type 'currencies' illustrates this.

This brings into focus two related architectural concerns relating to the ontology and agentology models (which are typically not considered) –

- Inter-relationships: when the system has agency, there is no choice but to include the agentology in the enterprise model; how this should be done? Should the agentology or ontology be modelled separately or together? And if separately in what order? More radically, if one has an agentology, is there a need for an ontology?
- Content allocation: given that there is a range of knowledge that can be represented using either a de se or de re perspective; how should this choice be made? What knowledge should be in one, what in the other?

While we have established that an agentology is essential for agentive systems, we have not done the same for an ontology. One can regard the deitic outskirts as neutral with regard to de se and de re perspectives as they appear the same in both. If one does, then the de se (agentology) models in the bank example can be regarded as de re free, which suggests that one could, at one extreme, have a pure agentology enterprise model with no ontology.

On the other hand, for non-agentive systems, such as pure reporting systems, there is no requirement for de se knowledge. In these cases, the agentology model is not required.

Given the growing scale and inter-connectedness of enterprise systems, interoperability and reuse (more specifically, reuse across agent-relative characters) are influential requirements. And with larger systems, as well as inter-system interoperability and reuse, there is intra-system interoperability and reuse to consider. We think these considerations should drive a preference for a de re approach, one that aims for a model closer to the other extreme, where the de se perspective is minimized as far as possible to the deitic centre - and the deitic

neighbourhood is modelled from a de re perspective. This is not to deny that, given the range of possible scenarios, there will be situations where a de se maximising approach makes sense, but there would have to be requirements that trump the need for interoperability and reuse.

The preference for agent-neutral forms does not always mean a binary choice. One could develop a single agent-neutral template model and use this to generate consistent, different agent-relative perspectives, simplifying interoperability. There will undoubtedly be cases similar to the three layer ANSI-SPARC model, where the core persisted data is in de re form and this is translated on-the-fly into a de se perspective for presentation to the users. The bank example illustrates how this might happen: the data could be persisted in simple de re correspondent accounts and translated into nostro and vostro accounts when required for specific users.

In greenfield development, the ontology is likely to have a wider breadth of reuse than the agent-relative agentology. This suggests a preference for building the ontology section of the enterprise model first and then the agentology section. So, both in terms of de se or de re preference and the order of construction, there are good pragmatic reasons for an 'Ontology First', or 'Ontology then Agentology' approach. Similar concerns apply to some types of brownfield projects, such as legacy system modernization.

# 5 FUTURE WORK

There are a couple of areas for immediate future work. Firstly, if one accepts that the agentology should be minimal, this raises the question of how minimal it can be. There are several precedents to follow. Lewis (Lewis, 1979), himself following a suggestion of Quine, considers possible worlds centred on a designated individual, or time-slice of an individual to characterize the deitic ‘now’. The idea is, as David Chalmers (Chalmers, 2006) puts it, ‘We can think of the centre of the world as representing the perspective of the speaker within the world’. In (Partridge, 1996) one of the authors introduces the system perspective and ‘dynamical’ (as their reference shifts) ‘now’ and ‘here’ event objects. This indicates that the deitic centre (I, here and now) is probably a reasonable base. Though as noted above, there may be a need to define derived de se perspectives (such as nostro and vostro accounts) to support user views.

Secondly, it makes sense to clearly differentiate the two perspectives in the model, which raises the question of the relations between the perspectives. There seems to be a need for a kind of identity mapping, such as that in Figure 2. What other kinds of mappings are needed? For example, can an object in the agentology be represented as an instance of a type in the ontology, and if so, how does this differ from the ontology-bound instantiation relation? These and similar questions need to be answered to provide a rigorous enterprise model.

# 6 SUMMARY AND CONCLUSIONS

The paper aimed to bring some clarity to requirements for de se and de re perspectives in enterprise models. As well as clarifying what these perspectives are – and that these two perspectives are distinct – it has provided good reasons for thinking that a typical (agentive) enterprise model cannot be just a de re perspective, that it needs to include a de se perspective as well. It has proposed good pragmatic reasons, based upon interoperability and reuse requirements, for an 'Ontology First', or 'Ontology then Agentology' approach both in terms of de se or de re preference and the order of construction.



# REFERENCES

Chalmers, D. (2006) 'The foundations of two-dimensional semantics', in Manuel García-Carpintero and Josep Macià (ed.) Two-dimensional semantics. New York: *Oxford University Press, pp. 55-140.*

Frege, G. (1997) 'Thought', in Beaney, M. (ed.) The Frege Reader. *Oxford: Blackwell, pp. 325-345.*

Holton, R. (2015) 'Primitive Self-Ascription: Lewis on the De Se', in Loewer, B. and Schaffer, J. (eds.) A Companion to David Lewis. *John Wiley & Sons, pp. 399.*

Kaplan, D. (1989) 'Demonstratives', in Joseph Almog, John Perry and Howard Wettstein (eds.) Themes From Kaplan. *Oxford University Press, pp. 481-563.*

Lewis, D. (1979) 'Attitudes de dicto and de se', *The philosophical review, 88(4), pp. 513-543.*

Milne, A.A. (1926) Winnie-the-Pooh. *London: Methuen.*

Object Management Group (2014) *MDA Guide rev. 2.0.*

Object Management Group (2003) *MDA Guide Version 1.0.1.*

ORMSC (ORMSC Draft) The MDA Foundation Model.

Partridge, C. (2002a) LADSEB-CNR - Technical report 05/02 - The Role of Ontology in Integrating

Semantically Heterogeneous Databases. *Padova, Italy: LADSEB CNR.*

Partridge, C. (2002b) 'The Role of Ontology in Semantic Integration', *Second International Workshop on Semantics of Enterprise Integration at OOPSLA 2002. Seattle.*

Partridge, C. (1996) Business Objects: Re - Engineering for Re - Use. *1st Edition edn. Oxford: Butterworth Heinemann.*

Partridge, C., Mitchell, A. and De Cesare, S. (2012) 'Guidelines for Developing Ontological Architectures in Modelling and Simulation', in Tolk, A. (ed.) Ontology, Epistemology, and Teleology for Modeling and Simulation: Philosophical Foundations for Intelligent M&S Applications. *First edn. Berlin, Heidelberg: Springer, pp. 27-57.*

Perry, J. (1979) 'The problem of the essential indexical', Noûs, 13(1), pp. 3-21.

Pressman, R.S. (2005) Software engineering: a practitioner's approach. *Palgrave Macmillan*.