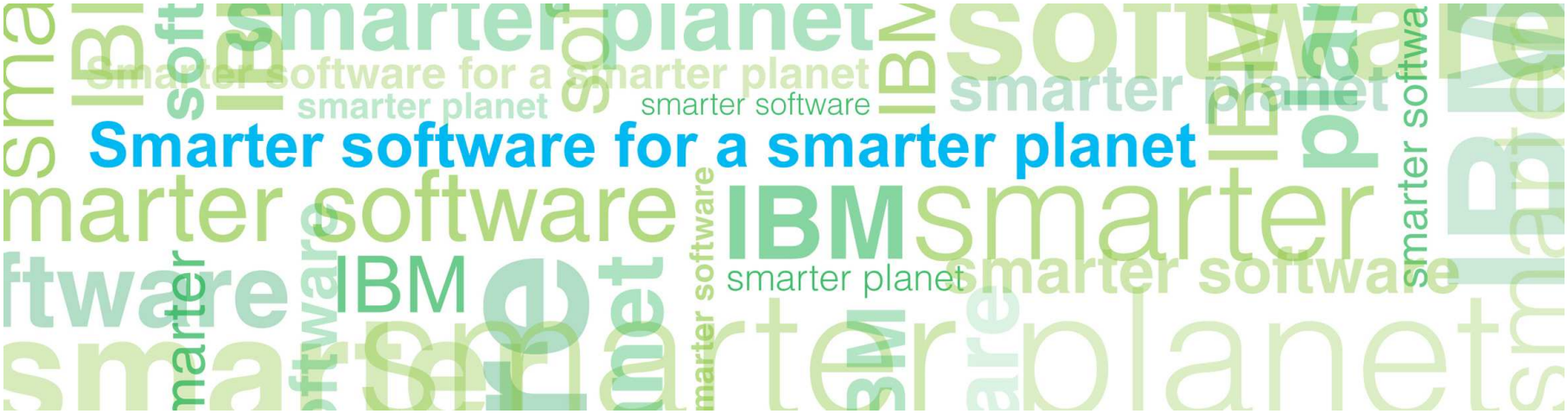
LanguageWare Resource Workbench 7.2

Data modeling



Introduction

- **Module overview**
  - Some thoughts around modeling
  - Some lessons learned from various customer/Demo work
  - It's not really a complete how to
  - May not even be best practice
  - Description of how they fit together and the logic behind them
  - Some questions to ask yourself when modeling to help with using these methods
- **Target audience:**
  - All audiences

Module objectives

After this module you will be able to:

- Start thinking about data modeling
- Start learning how to use LanguageWare Workbench to create annotators and data extraction models

Module roadmap

– **Data modeling**
  - What is modeling?
  - Some thoughts about modeling good practices and methods?
  - Some lessons learned from various customer/Demo work
  - Some questions to ask yourself when modeling to help with using these methods.

– **Summary and best practices**

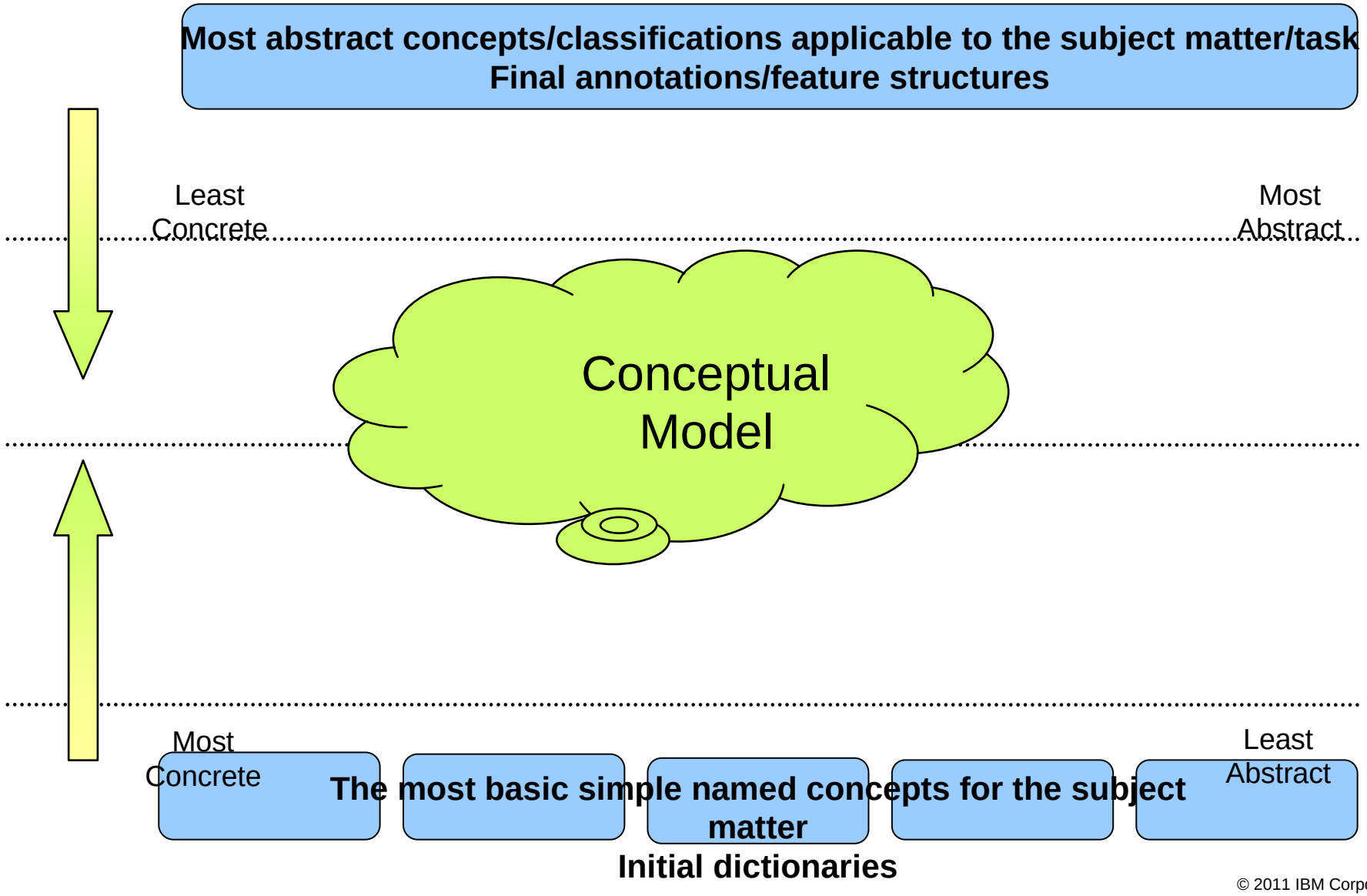
Before you start modeling

- LanguageWare is a very powerful, adaptable language processing engine
- BUT it is not a silver bullet that will solve every problem in one go
- Do not try to stretch a model to make it solve every problem. This may be possible, but hacks or over-fitting a model make maintainability a nightmare
- When modeling if you run into a brick wall ask yourself if the problem might be easier solved by pre/post processing…if so try to push it out to those stages

Where do you start?

- Start at both ends of the problem and work towards the middle
- Every subject matter/Text Analytics task has an abstract notion of how the information fits together
- Think about any Centralized Type-system, ontology or taxonomy that describes or defines what you want to extract
- Use this to define the start and end points of your model, i.e. the initial dictionaries, and the final annotations/feature structures you want, and what features they have/need
- Consider how the smallest functional elements of the model combine to form more meaningful units
- Consider how the abstract notions can be “broken apart” or are composed of smaller meaningful chunks

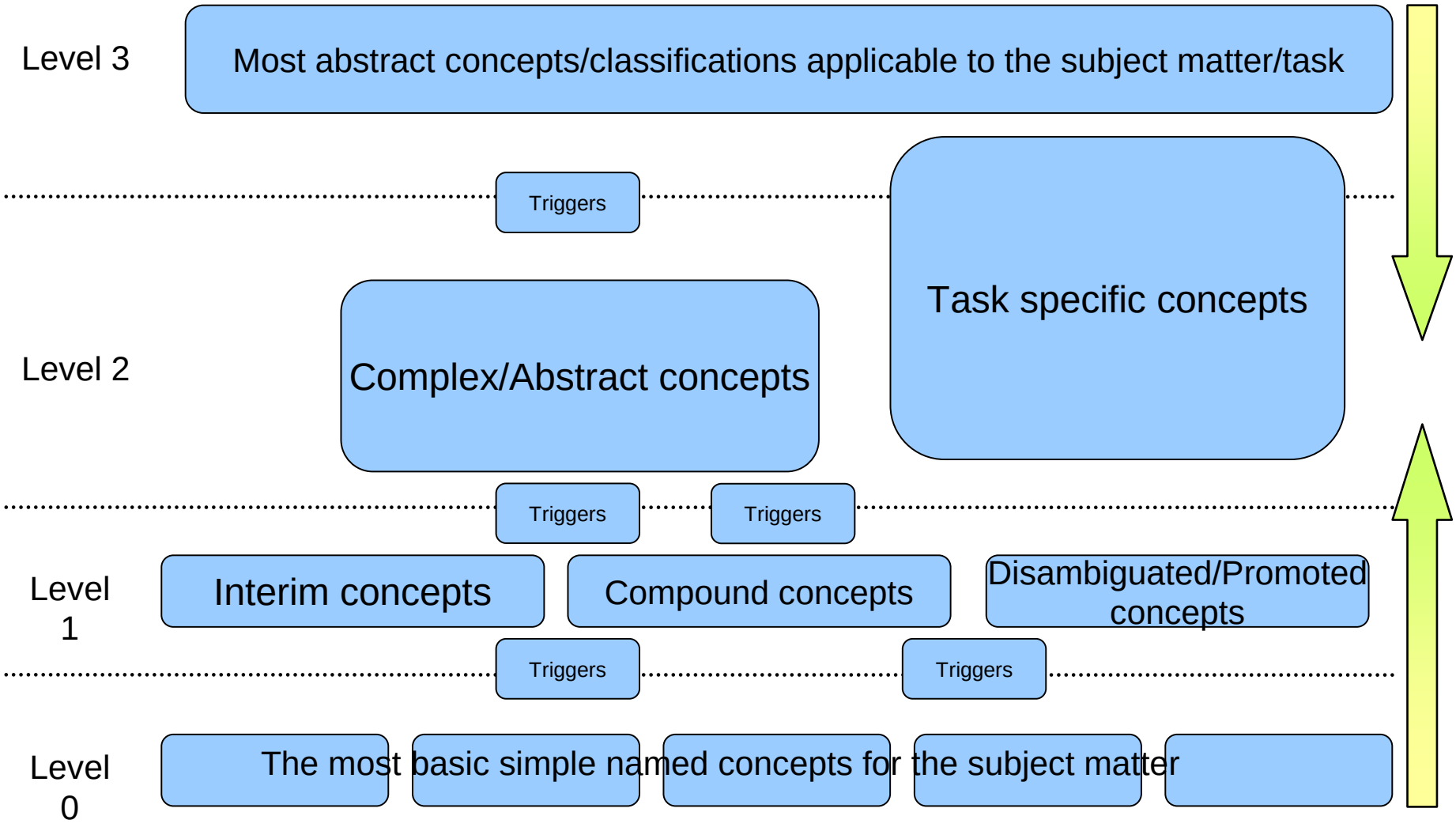
Where do you start?



Filling in the middle ground

- Once the end points are defined fleshing out the model is a matter of
  - Composing the simple (dictionary) elements to create more meaningful units using rules
  - Decomposing the abstracted end points to provide basic logic to drive rules development
- To move upwards in the model ask “how can I combine these things into something else?” and then write that rule
- To move downwards in the model ask “What is this made up of?” and then write that rule
- In both cases when deciding on the format of a rule ask “What sort of things are usually found near this?”
  - If those things are already defined in the model, rules logic will fill the gap
  - If not, additional trigger word/phrase dictionaries, or interim annotations might be needed
- This will not guarantee a complete model but gives a good structure which should be easily maintained and extended

The bigger picture



Types of dictionaries and rules

- Modeling in this way gives rise to 2 types of dictionary distinguished by both their usage and the type of “thing” they recognize
- Likewise 3 types of rule can be distinguished based on their function and the function of the annotations they create
- These rule and dictionary types are only general guidelines, but can be good for helping to separate and define the work done by the various resources

Two types of dictionaries

- **Type 1** – A list of things that are classified in the same way
  - Enumerable lists of “things”
  - Basic concepts with a semantic type (such as FirstName, Organization, Location..)
  - Eg. Town names, Car Manufacturers
  - Used predominantly at lower levels

- **Type 2** – *Stuff* that’s found in the vicinity of interesting *stuff*
  - Hard to classify as a single semantic type. Corresponds to the notion of a “trigger” or an “indicator”, that appears in the vicinity of important entities(example inc and & Co for companies)...
  - Anything from pronouns/prepositions to sub sentential units
  - Not always the same type of “thing”
  - Usually short lists
  - Eg. street, avenue, “was awarded to,” inc., corp.,
  - Used throughout a model for promotions, disambiguations and rule structure

Three types of rules

- **Type 1** – Disambiguation and promotion rules
  - Useful for confirming and overriding dictionary matches
  - Mapping into generic container types to drive more abstract rules
  - Create confirmed annotations
  - Used throughout the model but should become less frequent as things get more abstract
  - Annotations created by these rules are usually useful to more complex/abstract rules or to the end user/consumer
- **Type 2** – Rules that create something useful, an end type, a feature of an end type
  - Combining simple elements
  - Constructing complex annotations
  - Creating feature structures
  - Used at higher and lower levels of abstraction
  - Annotations created by these rules are usually useful to the end user/consumer
- **Type 3** – Rules which manipulate annotations creating task specific concepts
  - Combines simple and/or complex elements for a specific task
  - Driven more by the task at hand than semantics
  - Can provide insight that can be used to drive disambiguation/promotion rules, eg. spotting sections, lists
  - Used mainly in the middle ground
  - Annotations created by these rules are usually useful only to the model and just drive rules

Abstraction at the lowest level

- Proved very useful in customer engagements
- Can help reduce the number of dictionaries needed, hence reduces the rules workload…hopefully makes the whole model much simpler
- Keep similar things together, and ask “Can the distinctions be described as differing values of the same/similar features?”

Abstraction at the lower levels (Dictionaries) – Example

- **Example, we are interested in how people feel about the condition of a car in a Sentiment Analysis model.**
  - NegativeCarConditionSentimentIndicators: “dented” “scratched” “filthy” etc
  - PositiveCarConditionSentimentIndicators: “gleaming” “pristine” “like new” etc
  - Rules then have to work on both dictionary types
  - Introducing a new sentiment (indifference or unsafe) requires creating a new dictionary type and new rules to support it
- **Taking an abstracted approach to the data**
  - CarConditionIndicator has an attached sentiment feature: “dented[neg]” “gleaming[pos]” “scratched[neg]” “pristine[pos]” etc
  - Rules work on a single dictionary type and extract the feature which is passed up to inform higher level annotations
  - Introducing a new sentiment means simply adding the appropriate vocabulary to the dictionary with the associated sentiment feature value
  - Introducing a new feature (eg: interior or exterior) to add granularity to higher annotations is simply a matter of updating the dictionary and passing that feature up in rules, i.e. no new rules should be needed

Module roadmap

– **Data Modeling**
  - What is Modeling?
  - Some thoughts about Modeling good practices and methods?
  - Some lessons learned from various customer/Demo work
  - Some questions to ask yourself when modeling to help with using these methods.

– **Summary and best practices**

Summary and best practices

- The more abstract the concepts the better: Granularity can be achieved through features or through further annotations. This also allows you to write rules on “superclass” type annotations
- Features/Structures of higher annotations should be dictionary driven to as large an extent as possible: Updating the contents of complex annotations then becomes a matter of updating the dictionary
- If something needs to be a feature of an annotation you create in a rule, ask yourself can it be a feature associated with a dictionary type?: This makes it easier to abstract the problem, and also makes the features available as tests for rules
- Where multiple dictionaries are used to contribute to the same annotation (eg. Mesh/SnoMed Diseases) or where in context disambiguation is to be used (eg. negation), write rules on a container type into which the dictionary types are promoted/disambiguated using context and other triggers

Contacts

- If you have any questions, comments or suggestions, contact us using the LanguageWare email address *EMEALAN@ie.ibm.com* or on the developerWorks® Forum.

Trademarks, copyrights, and disclaimers

IBM, the IBM logo, ibm.com, developerWorks, and LanguageWare are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of other IBM trademarks is available on the web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml

Other company, product, or service names may be trademarks or service marks of others.

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON IBM'S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION. NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF ANY AGREEMENT OR LICENSE GOVERNING THE USE OF IBM PRODUCTS OR SOFTWARE.