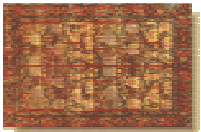


Grammar-oriented object design (GOOD): Writing Business Compilers

Ali Arsanjani

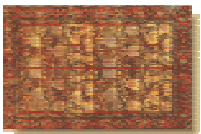
IBM

Maharishi University of
Management



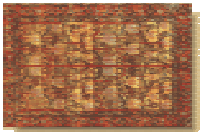
Let's Start with an Example

- A Student registration System at a University...
- Input: clicks on menu items and buttons, entering student and course information
- What is the format of the input?



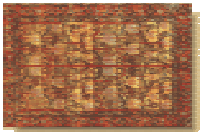
Adaptive Architectures

- Devising a system that is adaptive to frequent changes in requirements is an ongoing challenge in software engineering
 - And how I/T supports the business
- One of the ways to achieve this ability to adapt to new requirements is through re-configuration
 - Requirements change → re-configure the components



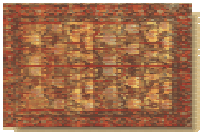
Adaptive Architectures

- In order to be able to reconfigure components we must first define and specify their composition; how the components are wired together
- This is often called an Architecture Description Language (ADL)
- Thus, we are looking for an ADL that allows us to represent not only the structure of a system (this is what ADLs do)
- But to define *three* elements necessary for a complete description of a business domain
 - Intention (Goals) , Function (Behavior) and Structure (how to construct)



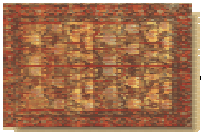
Business domains

- Business domains (BD)
 - telecommunications, banking, insurance, automotive, higher education, transportation, manufacturing, etc.
- Every BD has a language of doing business, a domain-specific language
 - Customer, Account, Wireless, Wireline, Product, Service Area, Airtime, Package, Usage, Equipment, etc.
- These terms are not used in an ad hoc fashion, they are composed into “sentences” that describe how business is done within each domain
- This business language can be represented as a *domain-specific language*

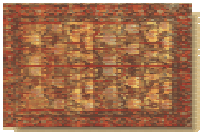


Domain-specific languages (DSL)

- Example DSLs
 - Hardware design, graphics packages, statistics, etc.
- We join ADLs and DSLs so we can build a business domain-specific language (BDSL) that can be used to describe a domain in terms of its components and their manners
- The way in which components should be combined from both static and dynamic aspects to achieve business goals can be described using a grammar
- This grammar defines
 - System structure (how components are composed and wired together)
 - System function (how components should behave in various contexts and circumstances)
 - System intention (goals)

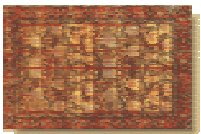


The application of Domain-specific Languages to Business Domains



Business Domain Language

- Thus, each business domain would have a generic language (DSL)
- ...that would be customized or re-configured
- ...when a new system is to be implemented or an older one changed to meet new requirements

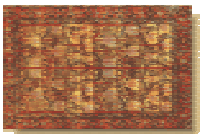


Business Systems

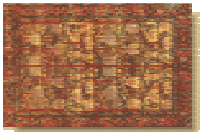
- ...are frequently built using object-technology these days
 - Java-based technologies or Visual Basic (e.g., .NET) based technologies
- Yes, they need to integrate with legacy too...
 - E.g., J2EE connector architecture
- But the key notion is that whatever they are built in (general purpose programming languages) they are far from the business language
- We need business domain-specific languages to represent key business concepts that allows business analysts working with software architects to construct reliable systems that meet business goals and objectives
 - 40-50 % of projects fail to meet business objectives

“Grammars + Objects”

- We combine grammars and domain-specific languages with
- Object-oriented technology
 - Analysis, design, programming languages
- Add more flavor in terms of component-based software engineering
- And get Grammar-oriented Object design (GOOD)
- $GOOD = (DSLs + ADLs) + (OO + CBSE) + \text{Business Modeling}$

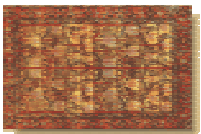


Every business domain has an inherent language that is waiting to be discovered



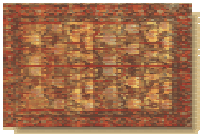
Grammar-oriented Object Design

- Grammar-oriented?
 - We need to describe (declarative) how a system of software components behave within a given context
 - Or, indeed how one software component behaves within a context (the principle of manners is applied recursively)
 - Thus, a context-free grammar can be used to describe a system's *manners or that of its components*.
 - *A system is a composition of software components*
 - *Running on a set of hardware nodes*



Key notions

- Manners
- Business Models
- Business Goals
- Large-grained Components
- DSL
- ADL
- MyLanguage as in “MyYahoo”
- Customize a language for a domain

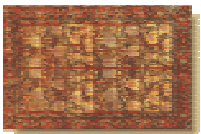


When is it justifiable to create a language
for a business domain?



To define business flow...

- To define how the components within the domain are to interact
- And how each component should internally behave
 - Abstract Specification for the internal design of component-based software architectures



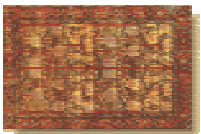
The Hows...

- How do you design components?
- How do you design the context?
- How do you design applications?
- How do you design variations?
- How do you compose applications?

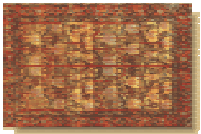


More important: Whole or parts?

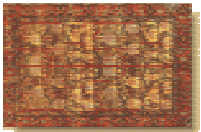
- The whole (assembly or composition or configuration of parts or components) is the key element, not the parts.
- Objects have
 - Identify, state and Behavior
- Define the whole by describing its
 - Services
 - Contracts
 - Manners
- Do you need variability? Are you
 - Building product lines?
 - Wanting to make rapid changes to meet new business requirements?
 - Rapidly introduce new products and services



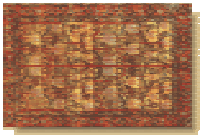
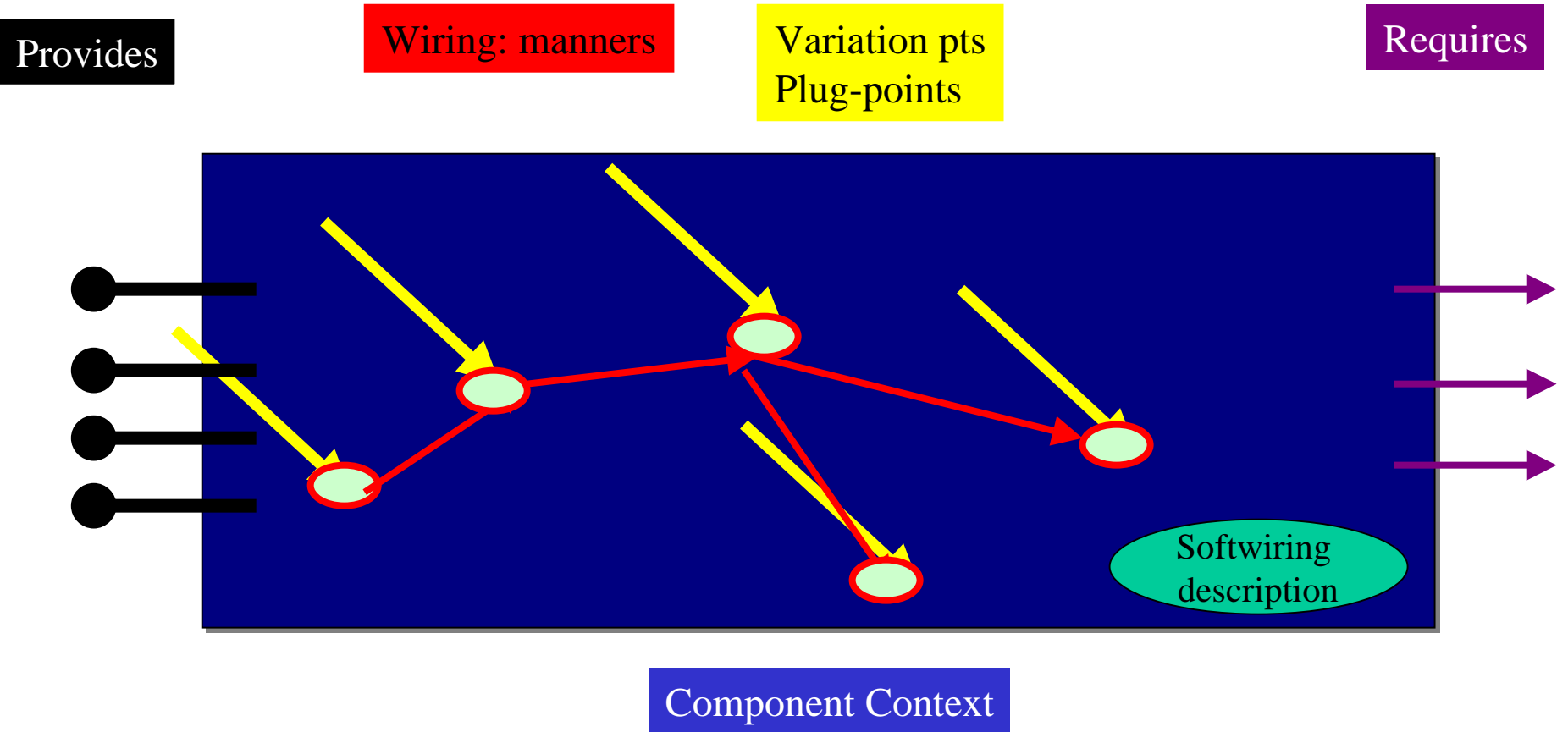
The configuration is more important than the components



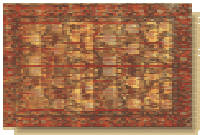
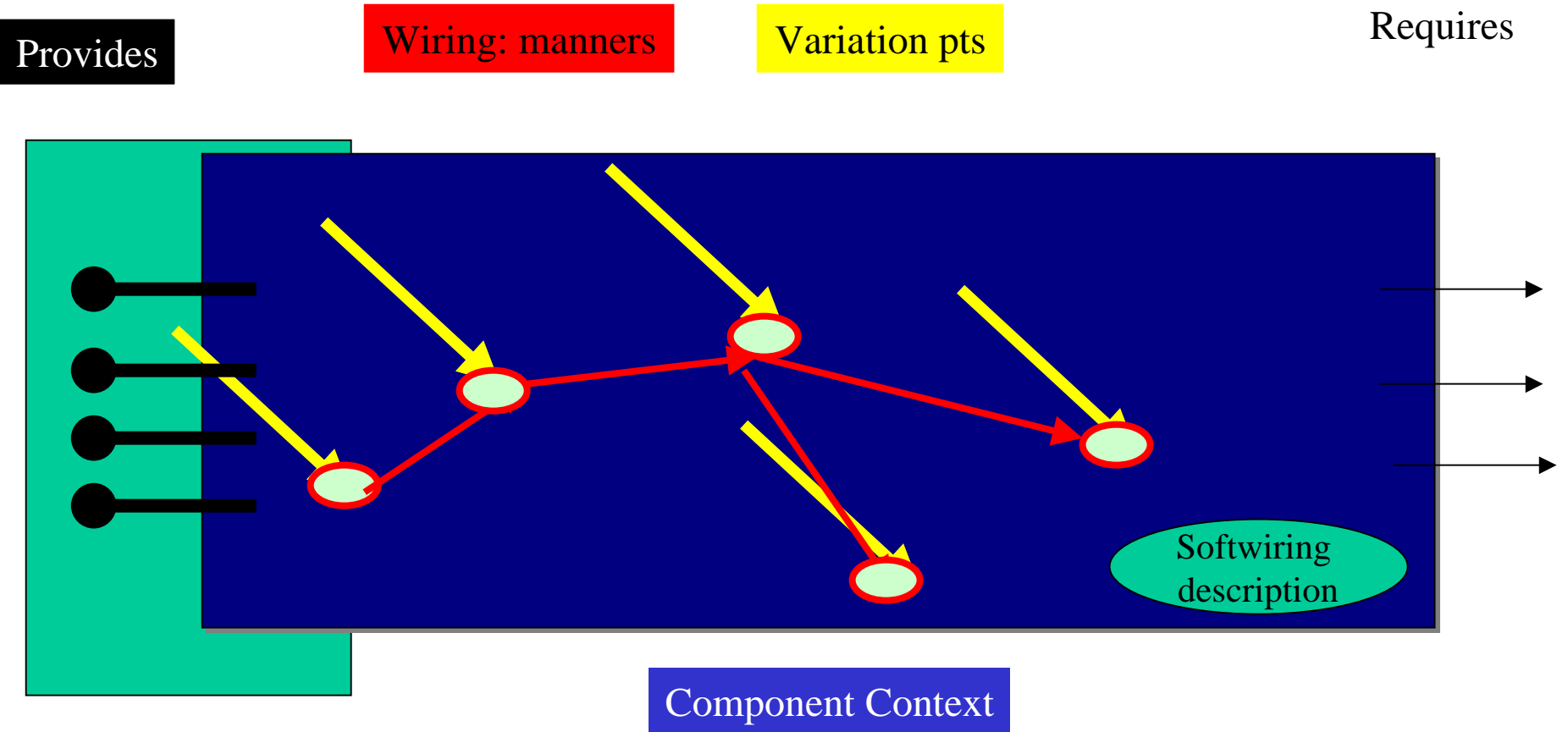
The Components of Component-based Development



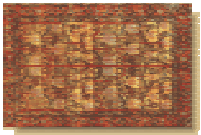
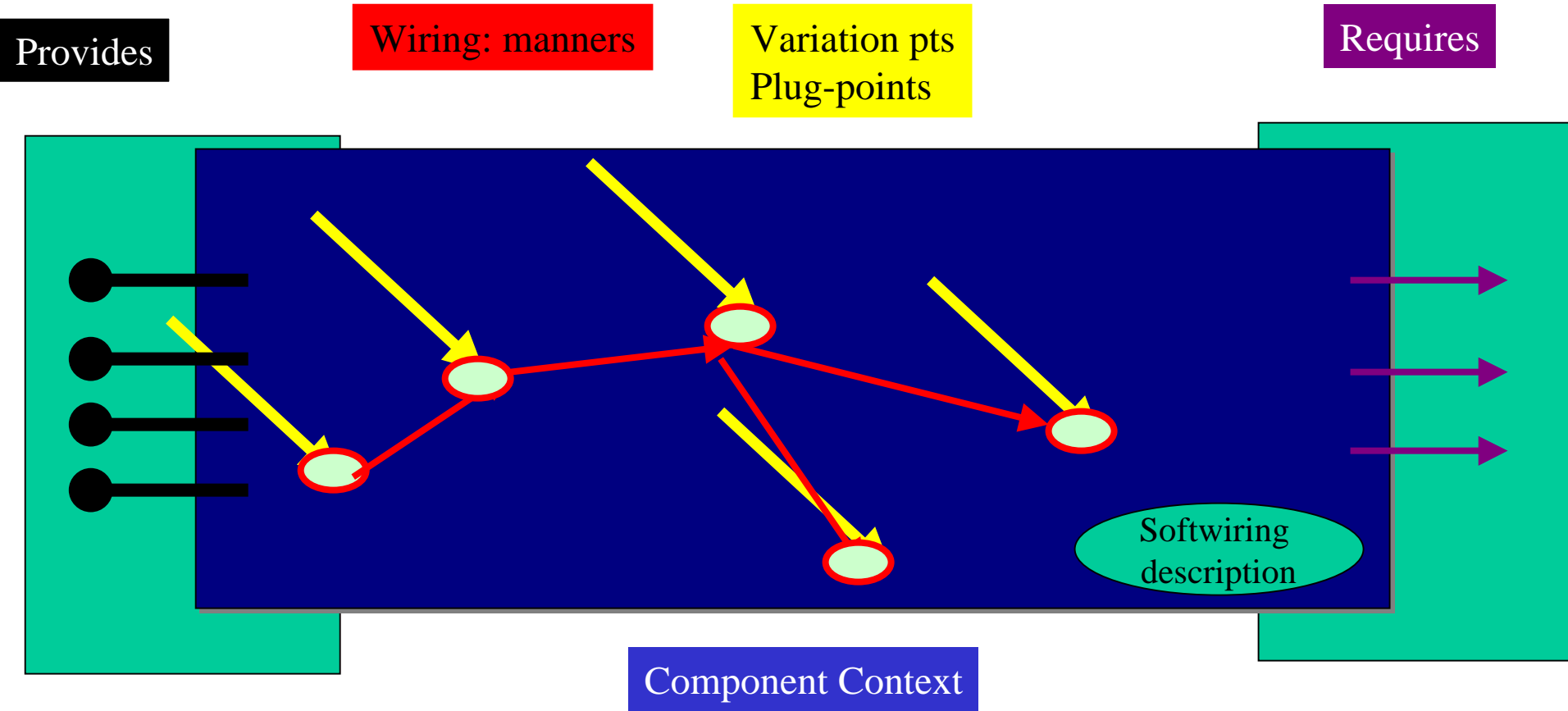
Components



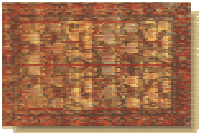
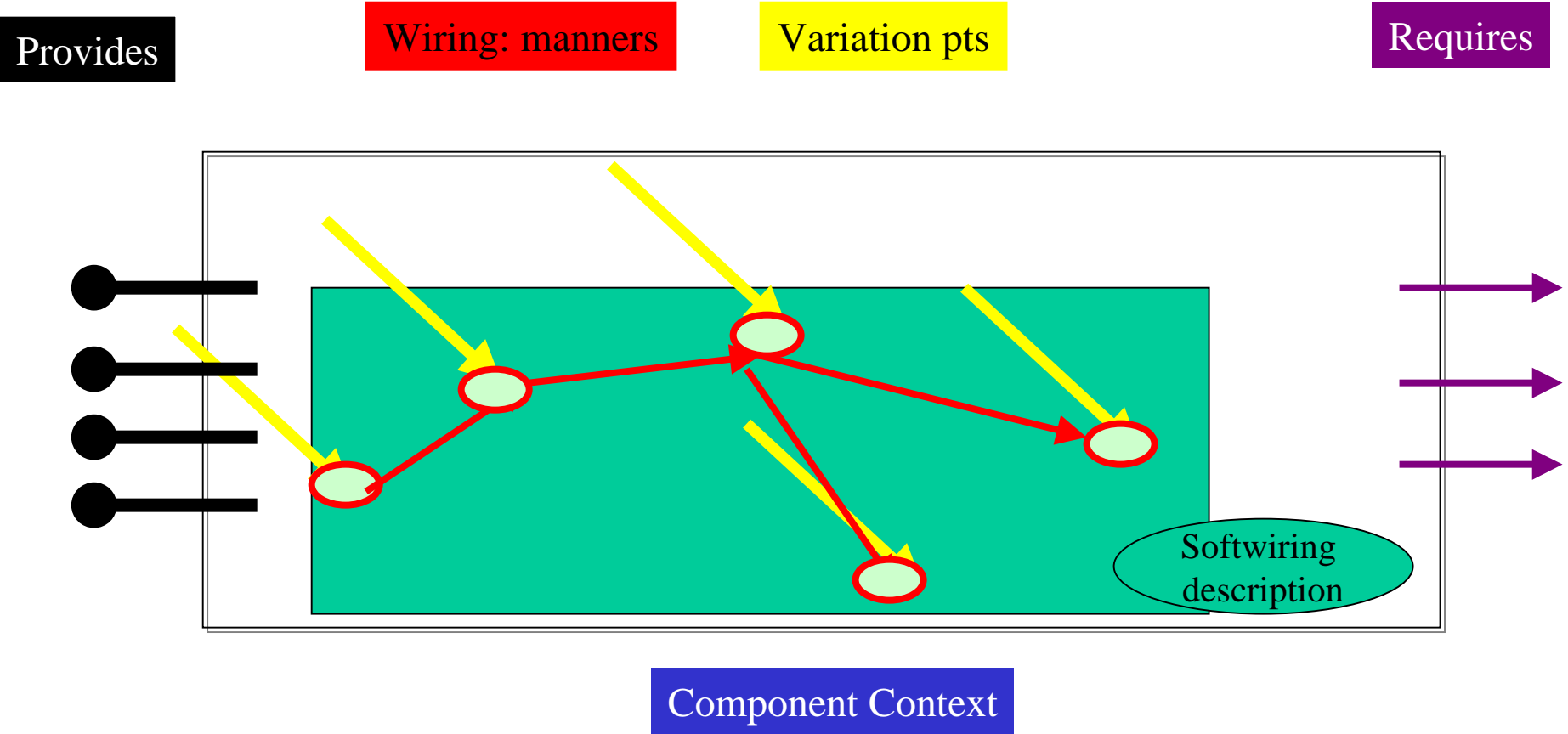
Services



contracts



manners



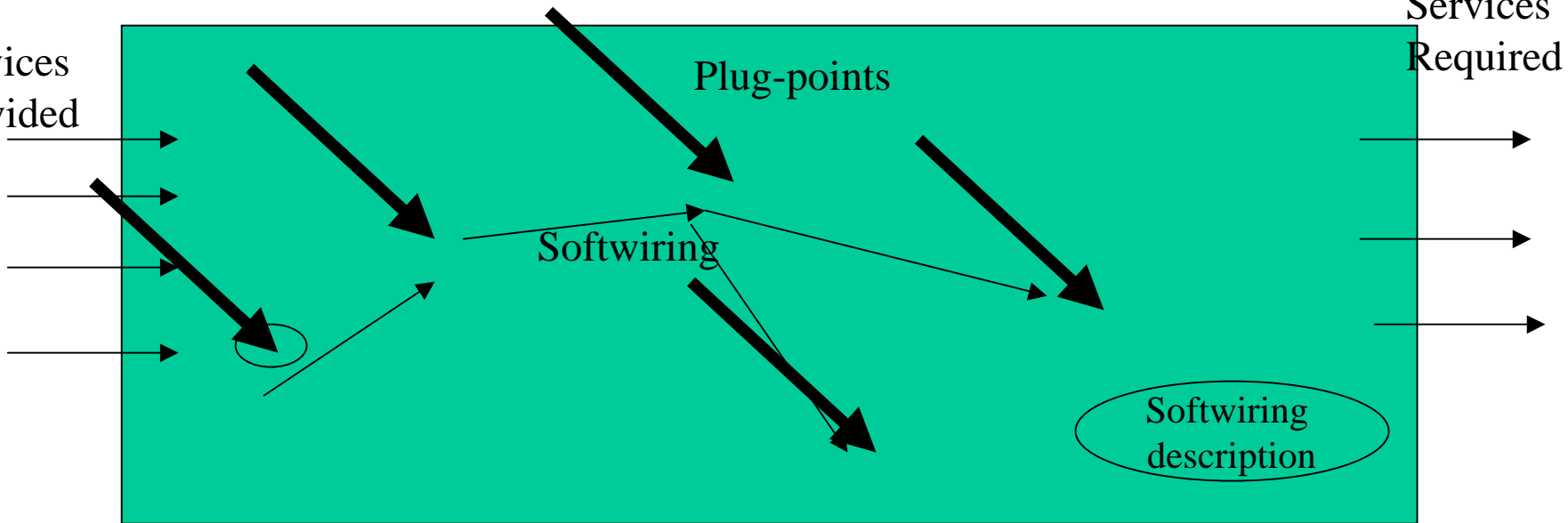
Context

External
Specification
(for consumer/
Assembler)

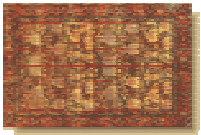
Internal Abstract
Specification (for builder)

Services
Required

Services
Provided



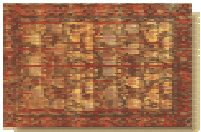
Component Context



Implementing manners using grammar-oriented object design

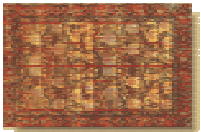
E-bazaar Case Study

www.arsanjani.org/e-bazaar/



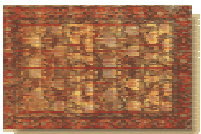
Business modeling creates a business architecture

- Helps identify
- goals (Goal Model)
- Conceptual model
- Process model
- Subsystem Analysis
 - Business processes
 - Subsystems
 - Components
 - Enterprise Components
 - Manners



Mapping the business architecture to a software architecture

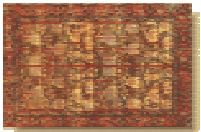
- Decreases the gap between business and I/T
- Although business goals and objectives are usually not communicated unambiguously...
- ...leading to incorrect assumptions when trying to realize the system
- We decrease this gap by
 - creating an isomorphic mapping between elements (business components defined as part) of the business architecture and a component-based software architecture
 - The business defines the boundaries of the business components
 - Business and IT jointly define the business domain language that is later refined by the software architect



Make Online Purchase

1.1 Use-case Grammar for Online Purchase use-case

**Online Purchase =
{[Identification], Presentation, Selection,
Purchase [Identification], Confirmation, Order
Fulfillment}**



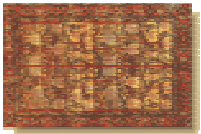
Make Online Purchase

1.1 Use-case Grammar

Online Purchase = {Identification, Catalog Selection, Selection, Purchase, Confirmation, Order Fulfillment}

Identification = {Challenge User with Login, Verify UserID and Password}

Catalog Selection = {Display Menu }



Make Online Purchase

1.1 Use-case Grammar

Online Purchase = {Identification, Catalog Selection, **Selection**, Purchase, Confirmation, Order Fulfillment}

Selection = {Browse Product Catalog, Select a Product Item, **Shopping Cart Operation**, {Selection | Quit}}

Shopping Cart Operation = { {Add Item to Shopping Cart | Delete Item From Shopping Cart | Save Shopping Cart | [Shopping Cart Operation] }, **Checkout** }

Checkout = {Complete Order Info}

Make Online Purchase

1.1 Use-case Grammar

Online Purchase = {Identification, Presentation, Selection, **Purchase, Confirmation, Order Fulfillment}**

****Purchase** = {Review Order, Review Terms of Agreement {Acknowledge Terms of Agreement, Submit Order | Cancel Order | Change Order to Quote}}.**



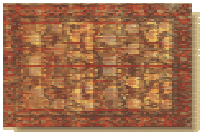
Make Online Purchase

1.1 Use-case Grammar

**Online Purchase = {Identification, Presentation,
Selection, Purchase,
Confirmation, Order Fulfillment}**

Confirmation = {Send confirmation number to user}

Order Fulfillment = {Pick and Ship Order}



Make Online Purchase Use-case Grammar

Online Purchase = {Identification, Presentation, Selection, Purchase, Confirmation, Order Fulfillment}

Identification = {Challenge User with Login, Verify UserID and Password}

Presentation = {Display Menu}

Assume Browse and Select has occurred

Shopping Cart Operation = { (Add Item to Shopping Cart | Delete Item From Shopping Cart | Save Shopping Cart | [Shopping Cart Operation]), Checkout }

Checkout = {Complete Order Info}

Complete Order Info = {{Verify Billing and Shipping Address| Select Billing and Shipping Addresses}, {Verify Shipping Method | Select Shipping Method}}.

Purchase = {Review Order, Review Terms of Agreement {Acknowledge Terms of Agreement, Submit Order | Cancel Order | Change Order to Quote}}.

Confirmation = {Send confirmation number to user}

Fulfillment is out of scope

Make Online Purchase Use-case Grammar

Online Purchase = {Identification, Presentation, Selection, Purchase, Confirmation, Order Fulfillment}

Identification = {Challenge User with Login, Verify UserID and Password}

Presentation = {Display Menu}

Selection = {Browse Product Catalog, Select a Product Item, Shopping Cart Operation, Selection}

Shopping Cart Operation = { {Add Item to Shopping Cart | Delete Item From Shopping Cart | Save Shopping Cart | [Shopping Cart Operation] }, Checkout }

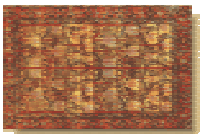
Checkout = {Complete Order Info}

Complete Order Info = {{Verify Billing and Shipping Address| Select Billing and Shipping Addresses}, {Verify Shipping Method | Select Shipping Method}}.

Purchase = {Review Order, Review Terms of Agreement {Acknowledge Terms of Agreement, Submit Order | Cancel Order | Change Order to Quote}}.

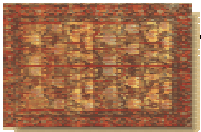
Confirmation = {Send confirmation number to user}

Order Fulfillment = {Pick and Ship Order}



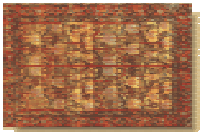
Summary: Common Problems Solved with GOOD

- Writing Business Compilers
- Adaptive and highly re-configurable Architectural Styles
- Reusable assets for each business domain
- Writing a formal, yet executable (!) description of the intent, structure and function of a business domain
 - Formal descriptions are often too narrowly focused and too difficult to develop to be executable and yet cover a significant portion of a domain.
- Non-intrusive changes can be applied to a higher degree
- Can jumpstart projects due to inherent, embedded domain knowledge
 - Requiring less upfront time in understanding the domain in greatest detail



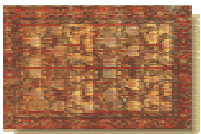
Summary of Steps in Using GOOD

- Write a grammar describing your domain
- Have the grammar drive the execution of flow of components within the domain
- Reconfigure the application by changing the grammar
 - Change or add production rules instead of changing code
 - Higher level of changeability



Main Points

- Veda is the language of Natural Law. The sequential unfoldment of creation is seen to arise out of the sequences of sound and silence in the Rig Ved.
- The impulses of creative intelligence are responsible for administrating creation
- Manners (GOOD) describe the business domain in terms of a domain-specific language (grammar)
- The system is an enfoldment of the business language
- Manage changes in the program through changes in the grammar



Exercise: Banking Account Management

