Challenges in Software Architecture

Programming models in a changing landscape

Dr. Axel Uhl

Chief Development Architect

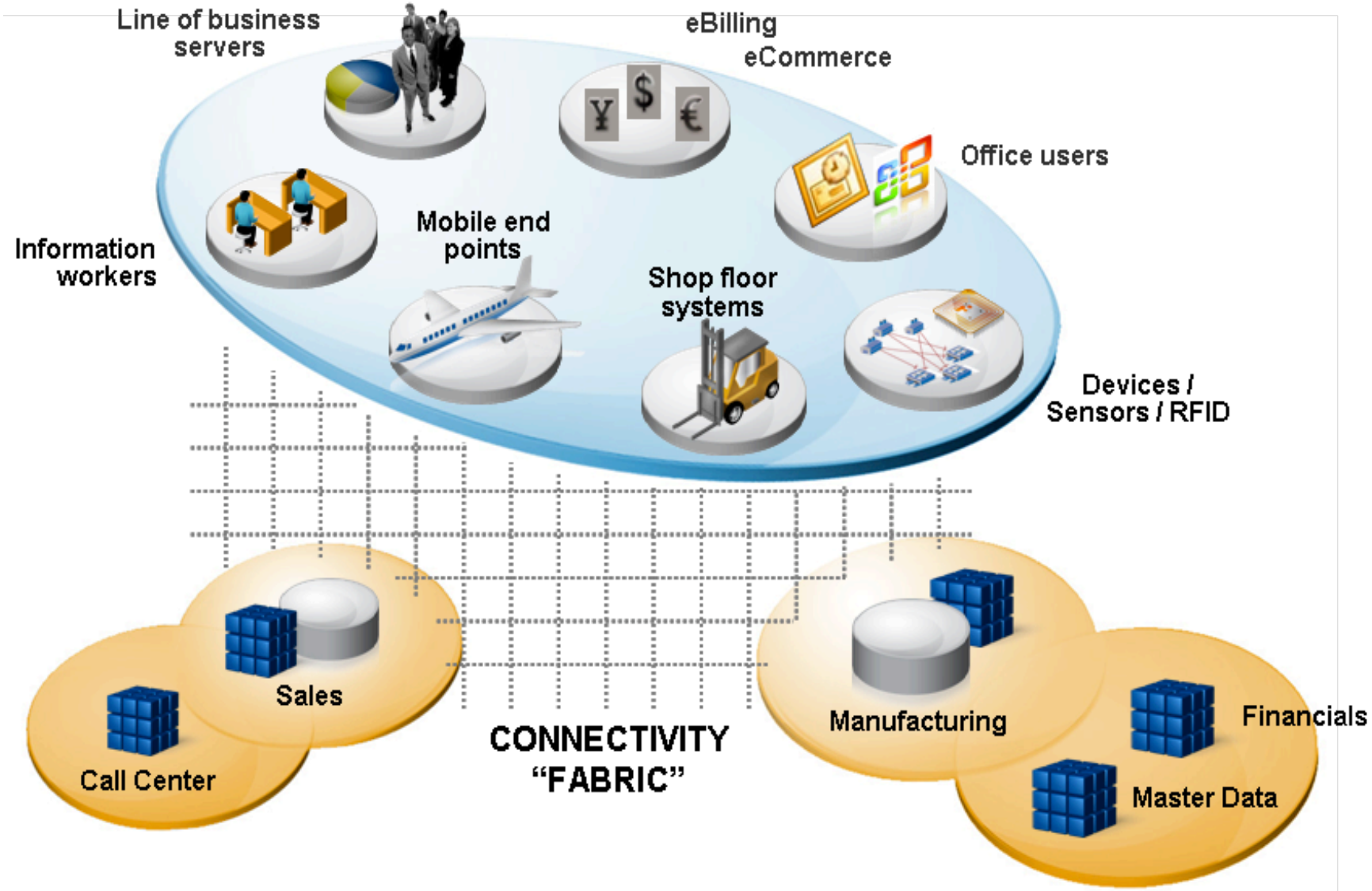
SAP, Office of the CTO

Agenda

- Challenging and Changing IT Landscapes
- Significant Optimization Potential in Our Stacks
- Next-Generation Programming Models
- Summary, Q&A

Emerging Business Landscape

Business-processes covering core and edge processes



The Web and Business Software Architecture

Business software architecture is once again shifting radically

- SOA is evolving to Webware (SaaS); there have been other shifts
- There are several underlying reasons for this shift
  - ubiquity, “flattening” (as in “flat world”), pervasive search, read-write web
- This shift impacts all major aspects of software
  - process, data, UI
  - development/programming models
  - delivery, governance, lifecycle management, …

But does this impact the fundamental underlying problems?

- ability to change, extend and integrate systems
- better automation
- lower costs of deployment and development, …

Global ↔ Local Delivery

Observations

- business processes being delivered from the “cloud”
- not limited to shareable/partitionable processes anymore
  - HR, Finance, Photoshop
- it’s really about
  - economics and dynamics of different delivery options, hardware and operations
  - ease of consumption, ubiquitous access
  - simpler and different revenue model(s), ad-financing
  - ability to customize, and share

Questions

- what are the governing dynamics of
  - local vs. remote access, change vs. share
  - interoperability, bandwidth, availability, security, usability and support
  - selecting the right partitioning (which processes to obtain from the “cloud”)
  - IT landscape management

Multi-Channel, Structured ↔ Unstructured

Observations

- applications used through various channels, each with special requirements and sometimes unique opportunities
  - mobile (very heterogeneous, semi-connectivity, location awareness, small form factor)
  - desktop (still heterogeneous, large form factor, rich feature set)
  - voice
- gaps in information workers' activities between structured and unstructured docs
- breaks in the consumption and provisioning of information
- limits our own ability to effectively collaborate

Questions

- How do we better collaborate?
- How can applications better support transitioning between structured / unstructured?

Information Gap

Observations

- Continued lack of semantics
- Missing integration of knowledge of user and context
- Different types of data have different search platform needs
  - Unstructured, structured, transactional, event, master
- Latency of real-time data
- Availability of great engines

Questions

- How do we best integrate a business user's context into search?
- How can tagging or universal ontologies/vocabularies help?
- What has b2b taught us on this?

Physical ↔ Digital

Observations

- RFID, sensor-networks, embedded systems enable more visibility.
- digital assets impact logistics, sales models and IP management.
- automated business processes that result from this are on the rise.
- need for real-world integration into business processes is already here.
- location awareness in cell phones and navigational systems as examples

Questions

- Is it about automation? Or is it about more data? Or its relevance?
- What parts of the infrastructure need to change to support this better?
  - analytics
  - automation
  - managing more data
- Does the nature of business activity change as a result of this?

“Flat World”

Observations

- Disaggregation of the value chain
  - over time, every activity that does not require presence will be done in a place that is more efficient economically
- Virtualization of enterprises
  - processes span organization / IT boundaries
  - visibility required transparently through these boundaries (think, e.g., GRC)
- Several governing factors
  - need for presence
  - infrastructure (network, communication latency, availability, …)
  - economics
  - automation vs. better delivery

Questions

- How will architectures support disaggregated value chains best?
- How different is multi-tenancy from per-customer visibility?

Adoption

Observations

- The modern web creates a massive “flattening” of information.
- Product adoption often lags information availability.
- This lag is much bigger in the business world than for consumers.
- Businesses need much smaller adoption and change cycles.
- Software services from different sources exhibit different lifecycles.
- But there are many aspects to this:
  - migration, training, integration, …

Questions

- How can we rethink change and flexibility in large scale software systems? Visibility?
- What are the major elements of the adoption lifecycle?
- What are the limiting factors? Knowledge transfer?

Agenda

Challenging and Changing IT Landscapes

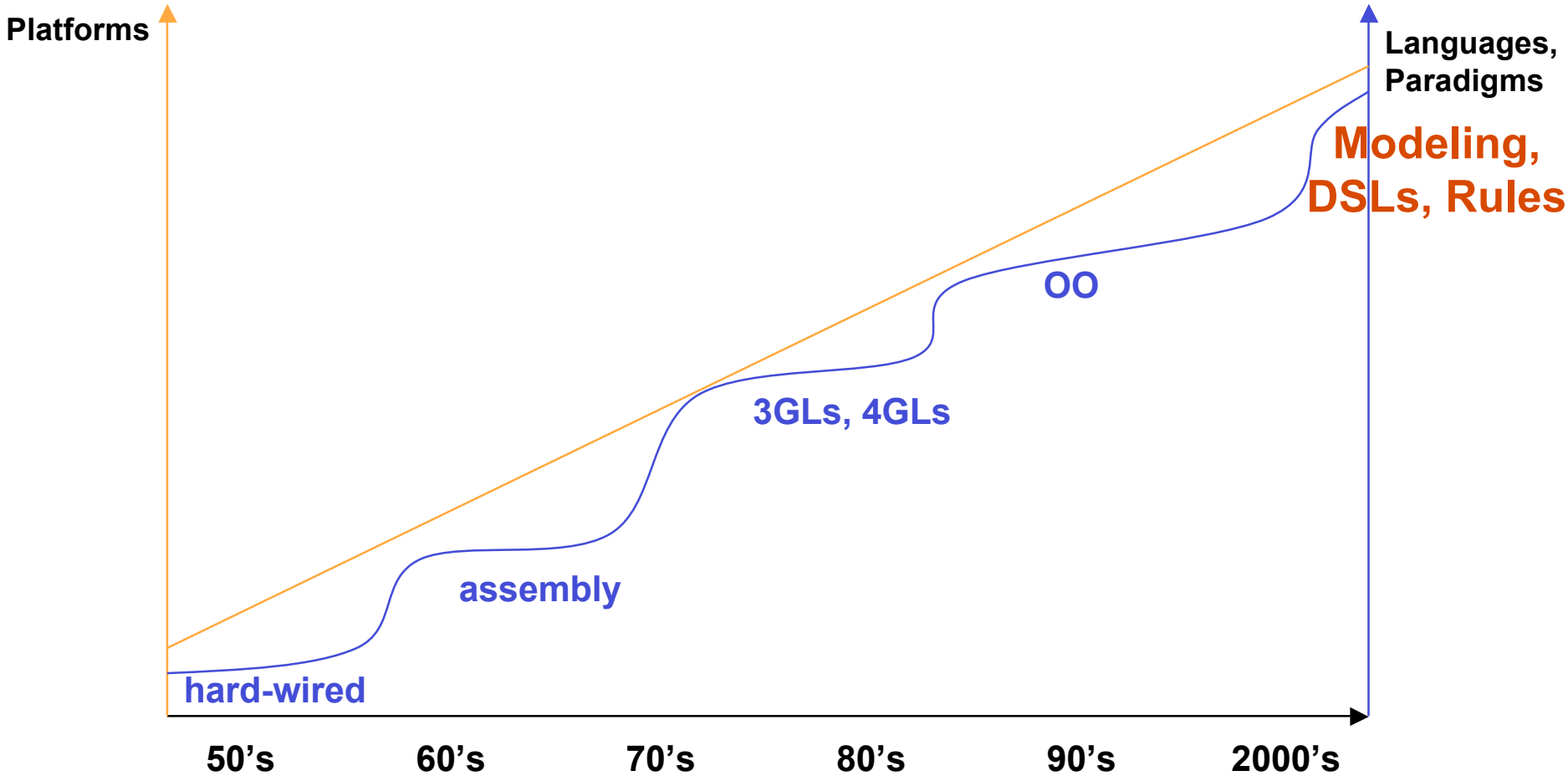
Significant Optimization Potential in Our Stacks

Next-Generation Programming Models

Summary, Q&A

“Moore’s Law” of Platform Complexity

Platform complexity “doubles” every few years.
Languages and Paradigms hardly keep up.



The Burdens of Re-Use

Observations

- The way we program is largely unchanged
  - some new thinking around AJAX
  - only early signs of web-specific programming languages and paradigms for easy development and change
- We are still largely creating new layers of abstraction:
  - each with their programming model, flexibility and purpose
  - benefits in isolation and separation of concerns, but
  - repurposing components in the stack into which they are assembled
  - overall performance and complexity of entire stack negatively affected
- I believe this choice is an artificial one. We can have both flexibility and optimization.

Questions

- What is a programming model that
  - maximizes development efficiency?
  - builds in reliability and performance benefits?
  - enables both benefits of abstraction and cost and cross-layer performance optimization?
  - can be used by a wide variety of developer types?

What do Programming Models have to do with it?

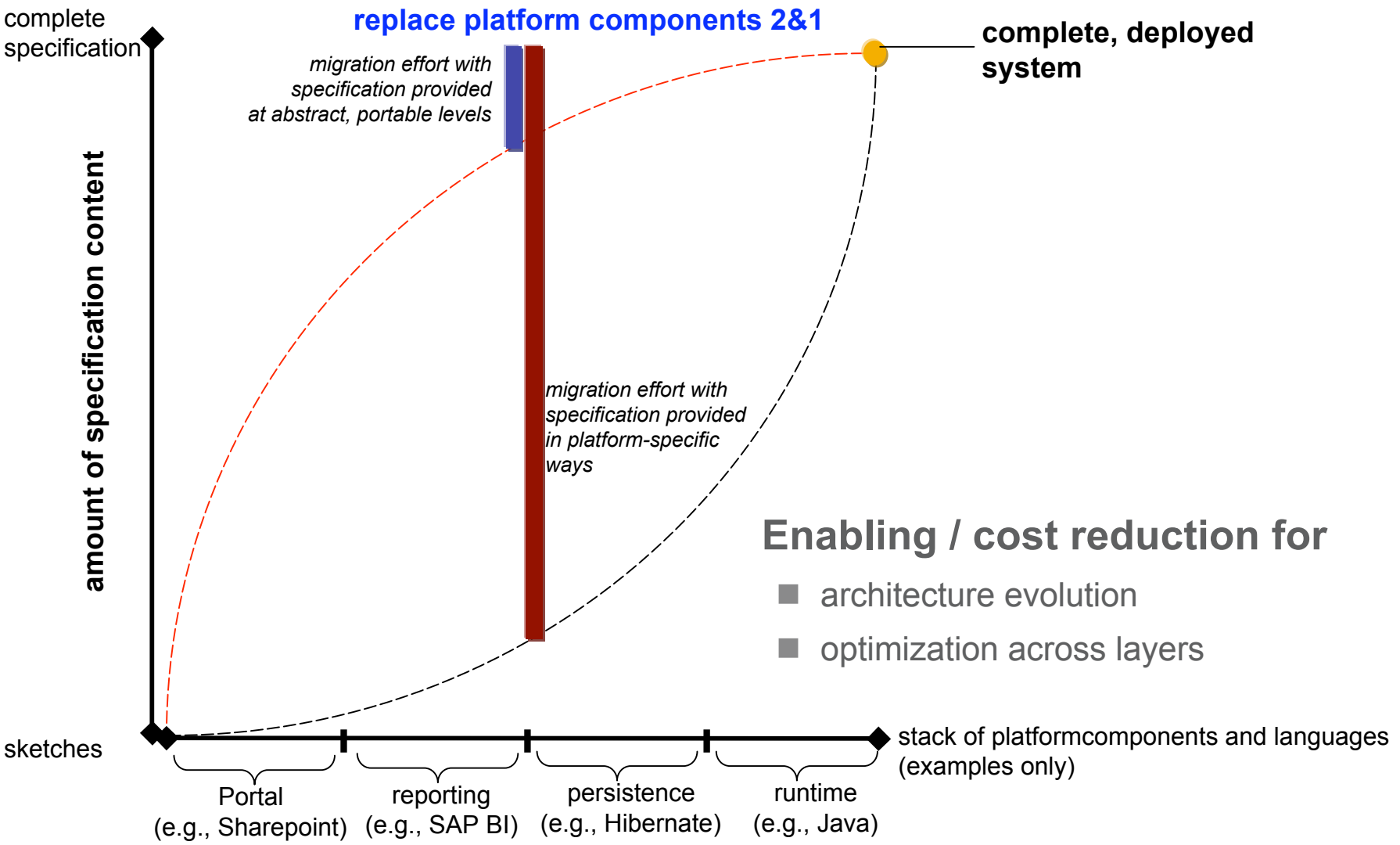
What is a programming model (PM) anyway?

- set of languages, frameworks/libraries, tools and guidelines

PMs should be used to cut complexity back to the essential complexity of the stack as it's being used or what it's been designed for.

- avoids unnecessary dependencies on specific elements of the stack
- leads to a greater flexibility in the evolution and optimization of the stack
- improves separation of concerns
- raises development efficiency

Improved Architecture Agility by Abstraction



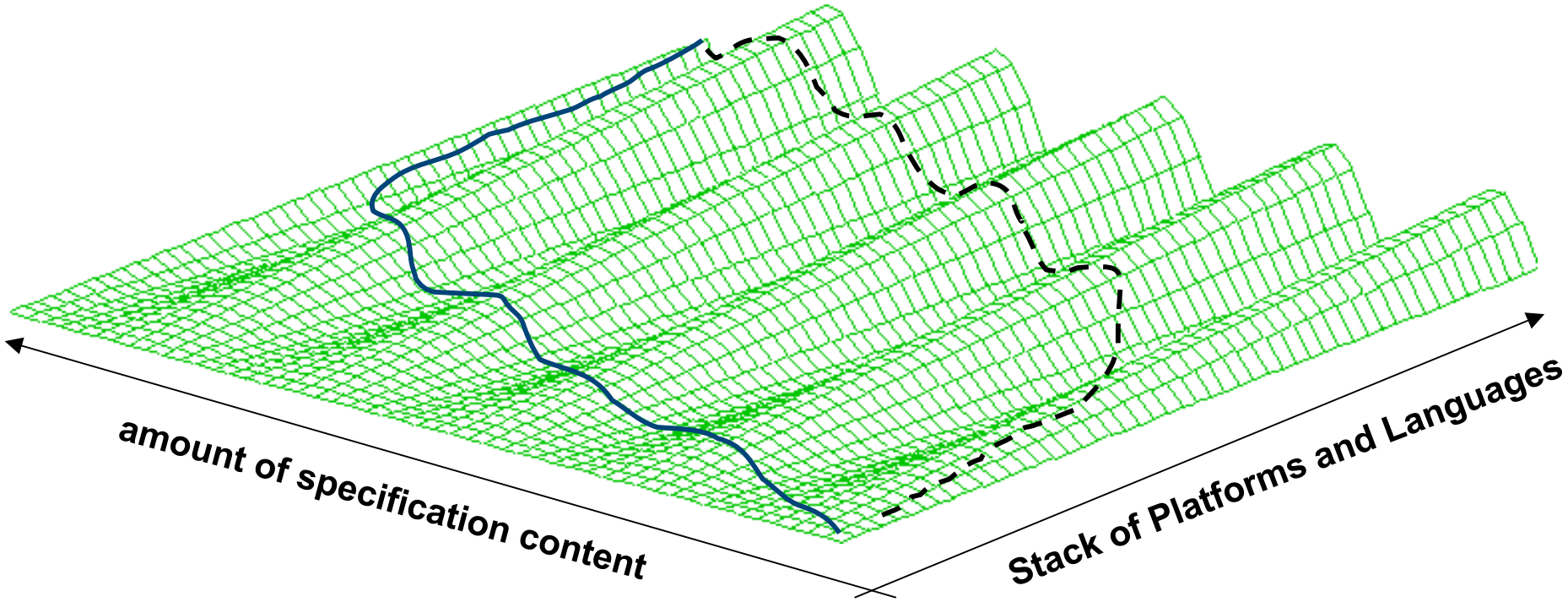
Enabling / cost reduction for

- architecture evolution
- optimization across layers

Improved Development Efficiency

Take path of least effort

- Detailing at low abstraction level causes extra effort and errors.
- Example: write an object-oriented business application in assembler



Agenda

Challenging and Changing IT Landscapes

Significant Optimization Potential in Our Stacks

Next-Generation Programming Models

Summary, Q&A

Demystifying Approaches to New PMs

“We could embed a DSL into a suitable host language.”

- Are tooling concerns addressed appropriately?
- How do you restrict the host language infrastructure to use only your DSL?

“Let’s build a new scripting language, and we’ll be doing fine.”

- But what distinguishes scripting in the first place?

“Ok, so we’re going to use a model-driven approach.”

- But what’s the difference between an executable model and a piece of code?
- And where is a graphical syntax more appropriate than an ASCII text?

Let’s take a closer look...

Scripting ↔ Non-Scripting

Scripting is about

- eliminating the compilation step
- using flexible type systems to make developer more productive

Blurring boundaries

- short compilation cycles for compiled languages
- JIT compilation (Java byte code → native; JSP to Java to byte code; ...)
- type system qualities (static vs. dynamic vs. duck typing; inference)
- memory management and bounds checking in compiled languages
- lifecycle management requirements for scripting solutions

Core values

- easy to learn
- making change easy
- good integration capabilities

Modeling ↔ Coding

There are many commonalities in what we call programming language and modeling language. Both

- have abstract and concrete syntax
- can be of rather declarative or imperative nature
- can use different types of representation
  (though we usually think of programming language artifacts as ASCII strings)
- strive for adequate abstractions, concern separation and aspect localization

Many issues of classical “programming” also exist for “modeling”

- physical partitioning of artifacts
- dependencies
- teamwork aspects (change management, versioning, ...)

What’s the difference between

- a code generator / model transformer and a compiler?
- a piece of C++ code and a sequence chart?

What's "*Modeling*?"

Herbert Stachowiak, *Allgemeine Modelltheorie*:

- Isomorphic representation
  - A model represents some *thing*.
  - Model and *thing* are connected by an isomorphism.
- Abstraction
  - The model suppresses irrelevant detail and focuses on important aspects.
- Pragmatics
  - The model is created for a purpose.

Examples of Models

- Crash test dummy
- London subway map
- UML model
- C# source code

Some parts get a lot more challenging in the "modeling" world...

Scaling to many Users

- >12000 developers at SAP
- some of our customers have larger teams than we do
- many different roles and skill sets

Scaling to many Languages / Metamodels

Different languages / language modules

- maintained by different groups
- with different release schedules

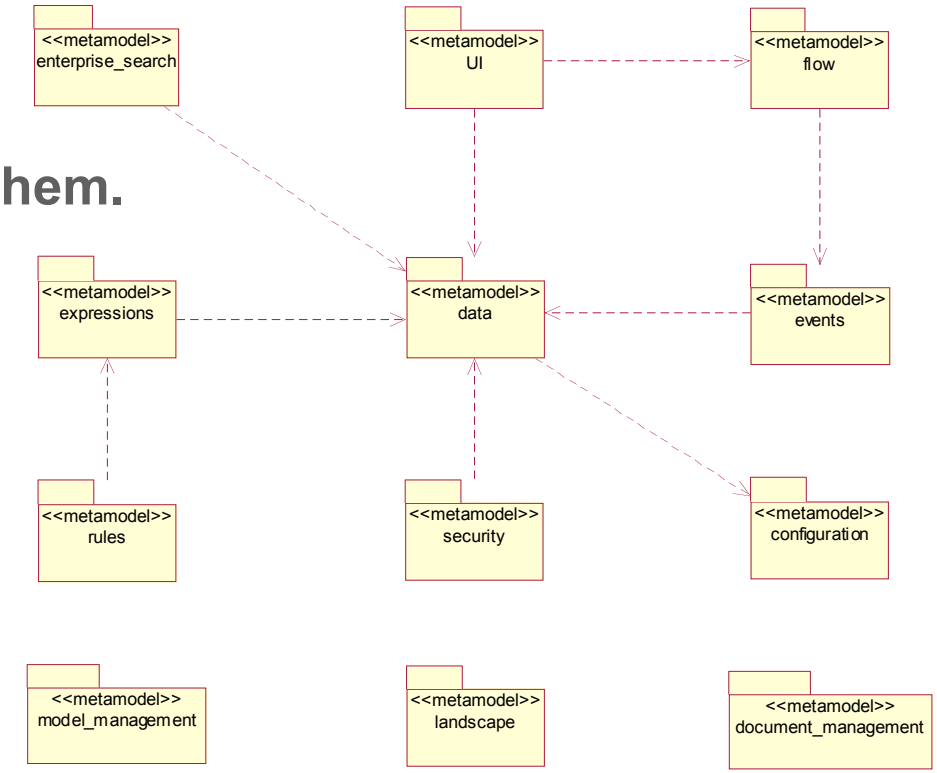
Yet, many links exist between them.

Individual users may cross boundaries

- avoid redundancies
- homogenize

Language may evolve independently

- allow for migration of artifacts



The Model Diff/Merge Problem

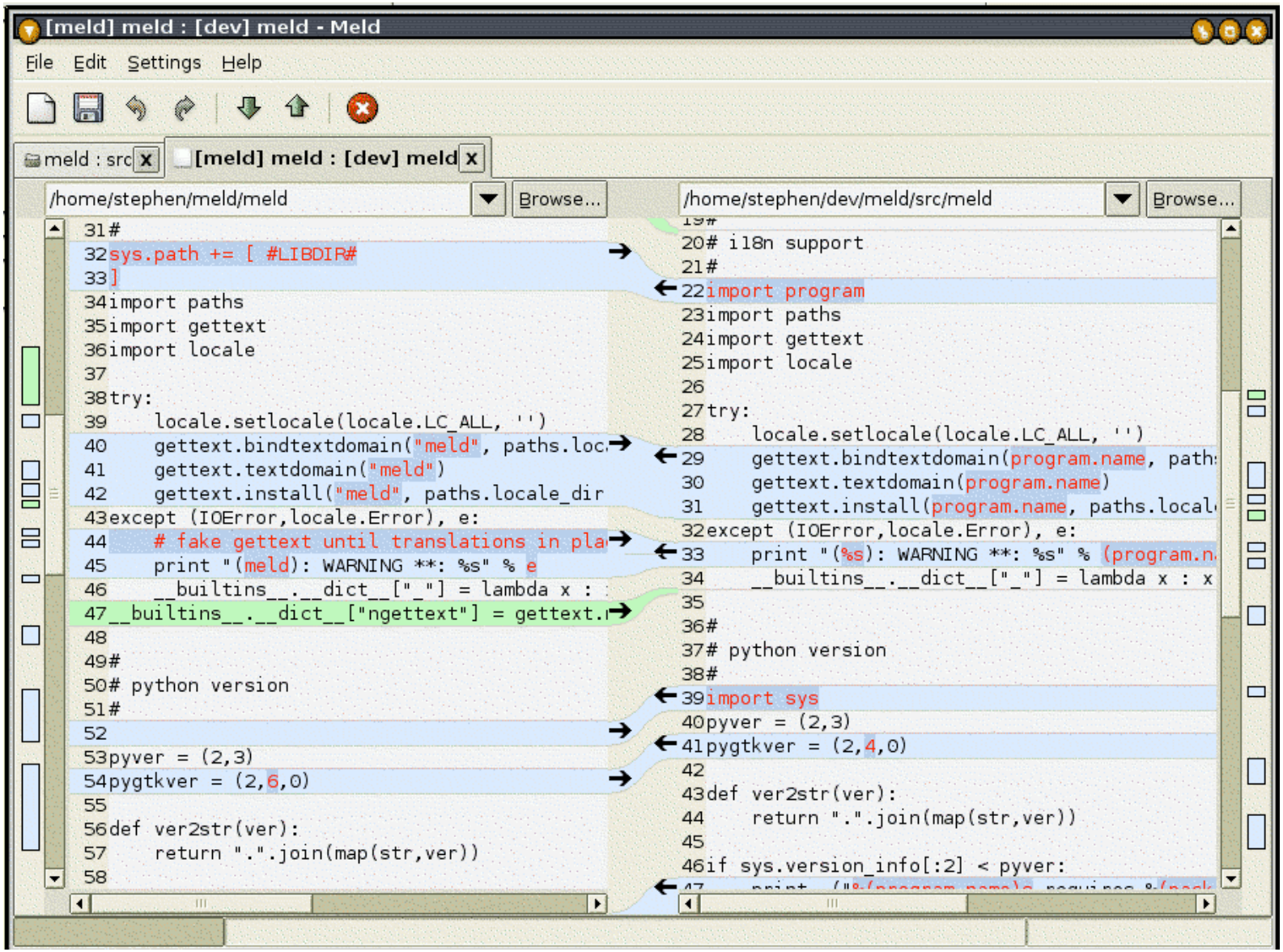
Logically atomic changes may affect multiple conflict/merge units, e.g.,

- cross-partition link addition/removal that is stored on both ends,
- delete propagation along composition hierarchy.
- Needs to be considered during merge operations.

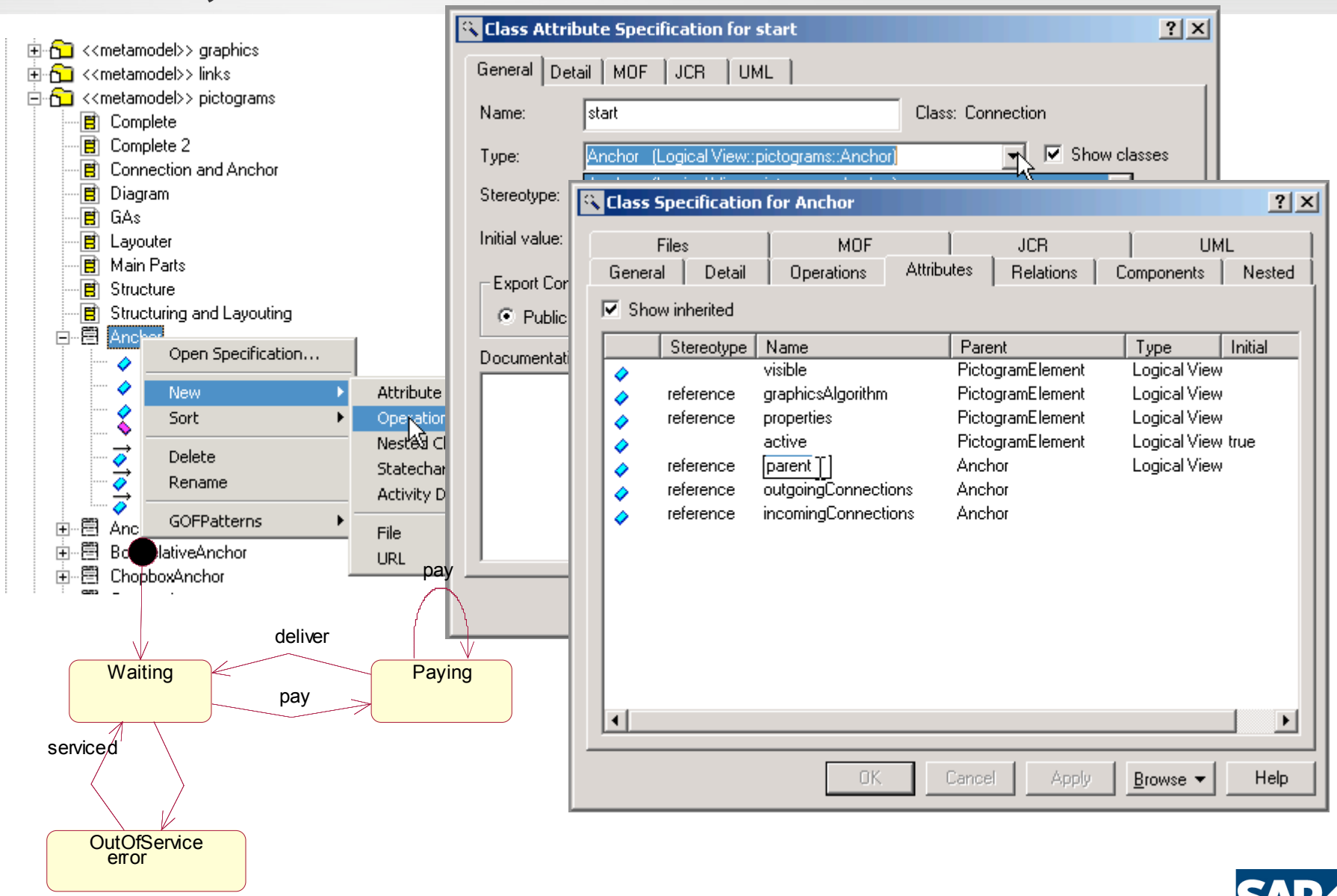
Structural differences between abstract and concrete syntax

How to display differences and merge conflicts of abstract model in concrete syntax (and in which)?

Understood for Text Syntaxes



Display the conflict in the tree, or in a form, and in which one?



Lifecycle Issues of Multiple (Graphical) Views

Changing a model through one view may update another

- views may be versioned and access-controlled artifacts
- extensions to models may be provided in multiple layers of the system

Changes in graphical views may be for viewing only...

- toggle expanded/collapsed setting on a diagram entity
- change the zoom level and panning position

...but should not necessitate checkout/versioning operation

- user may not have the permissions required
- creation of a new version not justified by minor changes of settings

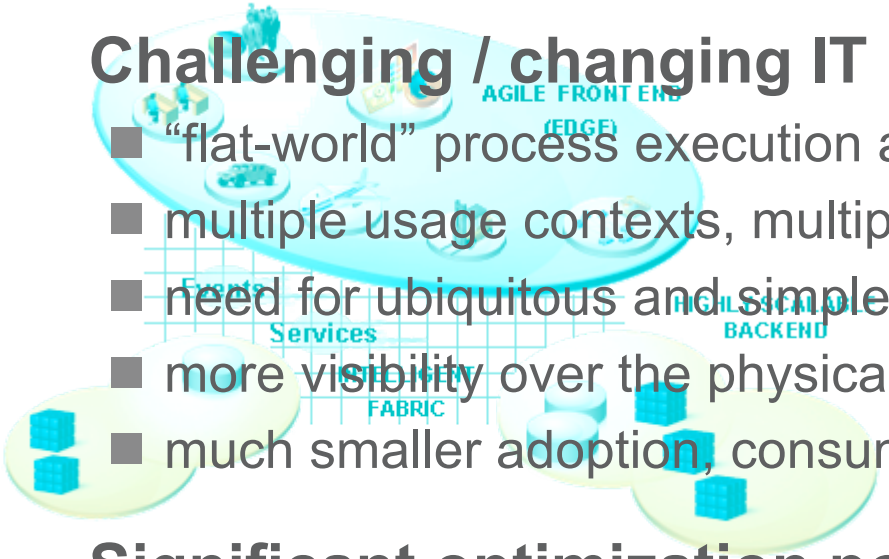
Research is only starting to understand

- e.g., Udo Kelter et al., SiDiff, http://pi.informatik.uni-siegen.de/sidiff/

Agenda

- Challenging and Changing IT Landscapes
- Significant Optimization Potential in Our Stacks
- Next-Generation Programming Models
- Summary, Q&A

Summary



Challenging / changing IT landscapes, most importantly

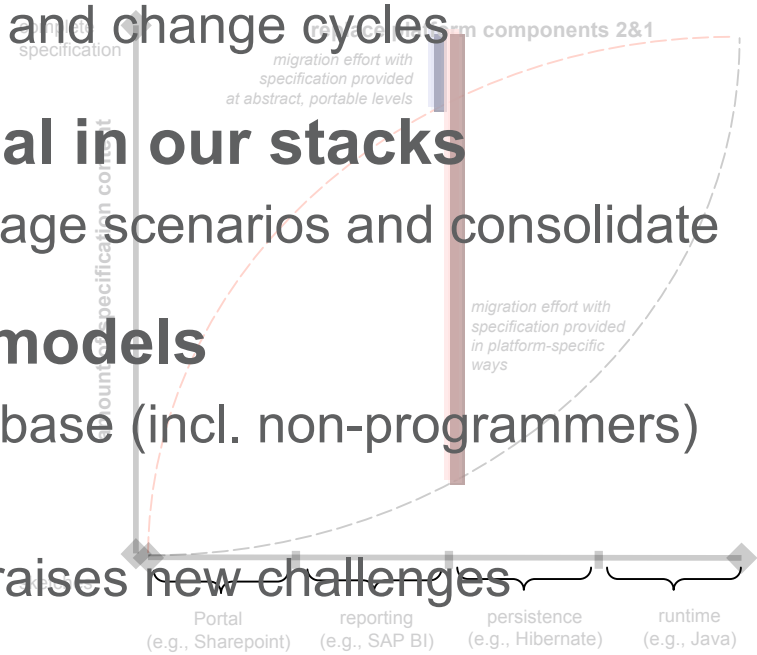
- "flat-world" process execution and consumption, SaaS
- multiple usage contexts, multiple access points, multiple form factors
- need for ubiquitous and simple search and information access
- more visibility over the physical world
- much smaller adoption, consumption and change cycles

Significant optimization potential in our stacks

- tune components towards specific usage scenarios and consolidate

Next-generation programming models

- extend development to broader user-base (incl. non-programmers)
- allow for cross-stack optimizations
- but "just modeling" isn't enough and raises new challenges



Thank You

Copyright 2007 SAP AG
All Rights Reserved