Integrating Software Architecture-Centric Methods into Object-Oriented Analysis and Design
Motivation (1)

- Software architecture is defined as “… the structure or structures of the system, which comprises software elements, the externally visible properties of those elements, and the relationships among them (Bass et al, 2003)”

- Given this definition, every software system has an architecture; and if one were to implement a system two different ways, each would have its own architecture
  - What makes one superior to the other?

- Suitability of an architecture is measured in terms of its fitness to purpose
  - The purpose of a system is often defined by its business goals and, therefore, it is critical that these business goals be considered during the architectural design
Motivation (2)

- Therefore, the choice of methodology for the development of the architecture for software systems has a direct effect on the suitability of that architecture.
- If the development process is driven by the user’s functional requirements, we would expect the architecture to appropriately reflect those requirements.
  - OOAD, for example, focuses heavily on functional requirements.
- The same phenomenon is true in development approaches that stress the importance of systemic quality attributes or other non-functional requirements.
  - Architecture-centric approaches, for example, provide explicit and methodical guidance to an architect in creating systems with desirable qualities and goals.
- An integrated approach that combines these two alternatives to capture the best of both “breeds” in a single architectural development process is desirable.
Motivation (3)

- Consider a company that builds hardware-based field systems for controlling a building’s internal functions, such as heating, ventilation, air conditioning, access, and safety. Hardware’s commoditization has led to shrinking profit margins. To improve the margins, the company wants to develop a software system to automatically monitor and control the building’s internal functions. The system users would be facilities managers, and the system would broadly perform the following functions:
  - Manage a network of hardware-based field systems used for controlling building functions.
  - Issue commands to configure the field systems.
  - Define rules on the basis of property values of field systems that trigger reactions and issue commands to reset these property values.
  - Trigger alarms notifying appropriate users of life-critical situations.

- The company wants to offer this product in new and emerging geographic markets and expand its sales channels by letting value-added resellers sell the software system under their own brands. The resellers would support field systems from manufacturers they choose.
Motivation (4)

• Looking at this example, it is clear the business goals set forth by the company would have a significant effect on the architecture of the building automation system without necessarily affecting its functionality
  – Considerations would have to be made to take language, culture and regulations of different markets into account
  – Hardware devices from many different manufacturers would need to be supported

• Tradeoffs would need to be made and risks assessed to determine the extent to which the product should support these goals
  – Depending on the company’s comfort level with the tradeoffs and risks, these goals may need to be refined, e.g. scaling back on the intended markets

• All of these business decisions require input from technical staff to determine the impact of such requirements and to inform the technical staff of the importance of these systemic requirements
Agenda

- **Goal**: Using an example create an architecture with OOAD and architecture centric approaches, understand their fundamental differences and the benefit of integrating the two approaches

- **OOAD** (Cheeseman and Daniels, 2001)
  - Requirements definition
  - Component specification
  - Component architecture

- **Architecture-centric methods** (Bass et al, 2003)
  - Quality attribute workshop (QAW)
  - Attribute driven design (ADD)

- **Integrated approach**
System used for illustration (1)

- MSLite system is a unified management station for the building automation domain that will automatically monitor and/or control the internal functions of buildings, such as heating, ventilation, air conditioning, lighting, access and safety
System used for illustration (2)

- The intended users of MSLite are facility managers who need to operate the many systems required to support building functions.
- Since there are a large number of these systems, a Field System Simulator (FSS) is used during software product development to simulate the building automation domain.
- An FSS configuration file is used to create the initial configurations of the simulated systems, including their structure and the initial values of their various properties.
- The business goals for this system are:
  - Achieve market expansion by selling it in new and emerging geographic markets with the accompanying demands on modifiability and interoperation.
  - Open new sales channels in the form of value added resellers (VARs).
OOAD Workflow (Cheeseman and Daniels, 2001)
Requirements workflow
Developing business processes (1)

- The business process on the right shows the building configuration workflow.
- This workflow is triggered when a facility manager receives a new building operation policy or an update to an existing policy.
- Depending on the policy details, a new rule may be created, an existing rule may be amended or no change may be required if the policy is already implemented by the existing rules.
- The rules can have different reactions such as issuing commands, generating alarms or sending notifications.
Developing business processes (2)

- This business process shows how alarms are handled by a facility manager.
- Depending on the alarm details, the facility manager may have to acknowledge an alarm, follow a Standard Operating Procedure (SOP) in response to the alarm or simply dismiss the alarm.
Identifying use cases

- Business processes can be used for identifying use cases.
- The subset of use cases and actors shown here comes from the analysis of some of the business process to be supported by MSLite including the ones described on previous slides.
- These use cases are grouped into four use case packages:
  - Use Case Package 100 (UCP100) is concerned with configuring the building operations.
  - UCP200 covers some aspects of monitoring of the building health.
  - UCP300 addresses the Personalization of the system by its operators (facility managers).
  - UCP400 manages the interaction with field systems.
- Use cases are goal oriented; in other words, at the end of a use case the actor initiating the given use case walks away from the system with something of value (Cockburn, 2000).
Developing business concept model

Business process descriptions introduce some significant concepts from the building automation domain.

These concepts are significant representing business entities that get created, destroyed, associated and used in various ways in order to achieve something of value.

Therefore, an important artifact in OOAD is a business concept model that captures these significant business terms and the relationships among them (Larman, 2004).

The business concept model of the MSLite system identifies a collection of domain entities and qualifies the way they interact or are related to each other using associations.
Component specification and architecture workflow
Specifying interfaces

- The use case model and the business concepts model serve as inputs for specifying the components for a system that become a basis for its architecture (Jacobson et al, 1999).
- In order to specify the components, we must first identify the interfaces they support.
Specifying system interfaces

- The use cases being at the boundary of a system help identify the system interfaces.
System components

- Once the interfaces have been identified, each interface (system or business) could be allocated to a single component or a single component could support multiple interfaces (this is where OOAD does not provide firm guidelines)
  - The allocation of interfaces is primarily driven by the principles of abstraction, encapsulation, and separation of concerns that result in loosely coupled and highly cohesive components.
  - While cohesion comes in many forms, the dominant form for most developers is that of functional closeness of the class members (Schach, 2006); one, therefore, is mainly performing functional decomposition of the system at this point with very little focus on its business goals or other quality attributes.
Business interfaces

• The process for obtaining these interfaces starts by refining the business concepts into core business types.
  - We identify core business types as business concepts in the Business Concept Model that have no mandatory associations.
  - A business interface is then created for each core type
Business components

- A component is created corresponding to each core type
Create a component architecture

Once the initial component specifications, their supported interfaces and their interface dependencies have been identified, a component specification architecture for a system can be created.

It should be noted that we have limited our analysis and design to a small fraction of the MSLite system. In reality, there are many more business and system components than those shown.
Focus of the OOAD approach

- The final architecture obtained in the previous section reflects the use case driven nature of OOAD.
  - The system components were created by aggregating functionally cohesive use cases.
- It also shows OOAD’s closeness to the model of the problem domain.
  - The business components were motivated by identifying core entities in the business concepts model.
- Although this approach does lead to architectures with loosely coupled and highly cohesive components that are easy to understand due to their semantic closeness to the problem domain, it is predominantly using functional decomposition with very little focus on the business goals or quality attributes.
Architecture-centric methods workflow
Architecture-centric approach

• In contrast to the OOAD approach, architecture-centric approach focuses on systemic properties that the software architecture must embody
  – Factors that influence the architecture, therefore, tend to be the quality attributes such as performance, modifiability, security and reliability (Bass et al, 2003).

• Consequently, these quality attribute requirements become a starting point for the architecture-centric methods

• Of course, these requirements must provide sufficient detail in order to be truly useful
  – For instance, it may not be sufficient to say that a system must be modifiable; any system is modifiable with respect to something, and a system can be modified with respect to any aspect given enough time and money – the question is modifiable with respect to what, when and with how much effort
Quality attribute workshop (QAW)

- Since they are the drivers for the architectural decisions, the first task is to determine the important systemic properties.
- This is done with Quality Attribute Workshop (Bachmann et al., 2002; Barbacci et al., 2000)
  - an architecture-centric method for eliciting quality attribute requirements from the stakeholders of a given system.
- The goal of this method is to establish a prioritized set of architecturally significant requirements in the form of quality attribute scenarios that are mapped to the business goals.
- Clearly, it is important these goals are known before the workshop can be conducted even if they are initially very general and will need subsequent refinement.
Business goals for MSLite

- **BG1**: In order to succeed in the Value Added Resellers market, the system must be able to support hardware devices from different manufacturers. This includes existing and to some extent future devices.
- **BG2**: It must be possible to modify the system to support different languages, cultures and regulations.
- **We can further refine BG2**
  - **BG2.1**: The system must allow changing all user interactions language to a language of choice. This includes languages with non-Latin characters and scripts written from right to left.
  - **BG2.2**: The field devices supported by the system can use different units. These units can be different from the units used by the user when specifying automation rules thresholds and commands. The system must be able to make all required conversions for rule evaluation and commands without errors and without user intervention.
  - **BG2.3**: Certain regulations and certifications require all life critical systems such as fire alarms and intrusion detection systems to operate within specific latency constraints. The system must be able to meet these latency requirements with a sufficient margin.
## Business goals and quality attributes

- The business goals are linked to quality attributes

<table>
<thead>
<tr>
<th>Business Goal</th>
<th>Quality Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BG 1</strong>: In order to succeed in the Value Added Resellers market, the system must be able to support hardware devices from different manufacturers. This includes existing and to some extent future devices.</td>
<td>Modifiability</td>
</tr>
<tr>
<td><strong>BG 2.1</strong>: The system must allow changing all user interactions language to a language of choice. This includes languages with non-Latin characters and scripts written from right to left.</td>
<td>Modifiability</td>
</tr>
<tr>
<td><strong>BG 2.2</strong>: The field devices supported by the system can use different units. These units can be different from the units used by the user when specifying automation rules thresholds and commands. The system must be able to make all required conversions for rule evaluation and commands without errors and without user intervention.</td>
<td>Modifiability</td>
</tr>
<tr>
<td><strong>BG 2.3</strong>: Certain regulations and certifications require all life critical systems such as fire alarms and intrusion detection systems to operate within specific latency constraints. The system must be able to meet these latency requirements with a sufficient margin.</td>
<td>Performance</td>
</tr>
</tbody>
</table>
Quality attribute characterization (1)

- For each quality attribute, the scenarios characterizing the corresponding quality attribute are described and prioritized with respect to business importance

<table>
<thead>
<tr>
<th>Quality Attribute</th>
<th>Attribute Characterization</th>
<th>Attribute Scenarios</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modifiability</td>
<td>Support for new Field System</td>
<td><strong>E1.</strong> Support for a new Field System offering functionality comparable to the Field System Simulator must be added. The configuration information and details of the interface (calling conventions, method names, etc.) are in a different format. A team of 2 developers reasonably experienced with C# extends MSLite to support the new system in 320 person hours (40 h per week and person, 4 weeks).</td>
<td>H</td>
</tr>
<tr>
<td>International Language</td>
<td>Support for International Language</td>
<td><strong>E2.</strong> A new language needs to be supported by the system. No code modification is required. A developer reasonably familiar with the system is able to package a version of the system with the new language in 80 person hours (40 h per week and person, 2 weeks) excluding string translation time.</td>
<td>M</td>
</tr>
<tr>
<td>Non Standard Units Support</td>
<td>Support for a new Field Device System</td>
<td><strong>E3.</strong> A new Field Device System using non-SI units is connected to the system. A system administrator configures the system to handle the new units in less than 3 hours.</td>
<td>H</td>
</tr>
</tbody>
</table>
## Quality attribute characterization (2)

<table>
<thead>
<tr>
<th>Quality Attribute</th>
<th>Attribute Characterization</th>
<th>Attribute Scenarios</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Performance</strong></td>
<td>Latency of event propagation</td>
<td><strong>P1.</strong> A field system detects a change of a property value and notifies MSLite. The system operates under normal conditions(^1). The value is updated on all user screens that currently display the property value within 3 seconds. The time durations specified in this scenario are performance goals and not hard deadlines.</td>
<td>H</td>
</tr>
<tr>
<td></td>
<td>Latency of alarm propagation</td>
<td><strong>P2.</strong> An event which should trigger an alarm is generated in a field device. The system operates under normal conditions(^2). The alarm is displayed on the user interfaces of all users that must receive the alarm within 3 seconds after the generation of the event.</td>
<td>H</td>
</tr>
</tbody>
</table>

\(^1\) \(^2\) Normal conditions must be specified such that quality attribute scenarios are measurable
Attribute driven design (ADD)

- After the architecturally significant requirements have been elicited, the architecture that meets these requirements is elaborated using attribute driven design (ADD) approach (Bass et al., 2003)
- The architecture elaboration approach we use as part of ADD is an iterative process
  - Starting with a monolithic component cumulating responsibilities for all the system functionality, architectural patterns and tactics are successively applied to satisfy each quality attribute requirement
- Very frequently, applying multiple patterns and tactics implies taking conflicting design decisions
  - This is why the end result of the architecture elaboration process is a compromise directly reflecting the quality attributes prioritization
Step 1: Confirm there is sufficient requirements information

The architecture of the system is designed in terms of the following three architecture requirements:

- **Functional requirements**: The users of the MSLite system would be facilities managers and the intended system would broadly perform the following functions:
  - Manage a network of hardware-based field systems that either currently used or might be added in future for controlling building functions
  - Issue commands to configure the field systems
  - Define rules based on property values of field systems that trigger reactions and issue commands to reset these property values
  - For life critical situations, trigger alarms notifying appropriate users

- **Design constraints**: The following constraints are required for MSLite:
  - **Concurrent sessions**: it shall support at least 15 concurrent sessions
  - **Field systems**: it shall support at least 30 field systems
  - **Events frequency**: it shall be able to handle at least 20 change of property values events from a field system per minute
  - **Automation rules**: it shall be able to handle evaluation of at least 50 automation rules per minute

- **Quality attribute requirements**: these correspond to quality attribute scenarios described in the tables on slides 28 and 29
Step 2: Choose an element of the system to decompose (1)

- The figure on the right shows the overview of the MSLite system architecture.
- Initially, the system consists of a single component, called the MSLite Server, responsible for all the functionality to be implemented.
- This is chosen for further decomposition.
- The functional requirements, design constraints and quality attribute requirements mentioned on the previous slide become the input for creating its architecture.

MSLite Server

Responsibilities:
- Send commands to FSS
- Receive Events from FSS
- Perform semantic translation for FSS data
- Route data to appropriate FSS
- Evaluate and execute automation rules
- Send automation commands
- Generate alarm notifications
- Display device data
- Capture/relay user commands
- Display alarm notifications
- Edit/Create automation rules
- Retrieve object data from FSS
- Store FSS Configuration
- Propagate Change of Value notifications
- Authenticate and authorize users
- Persist automation rules, user preferences, alarms
Step 2: Choose an element of the system to decompose (2)

- The key here shows the meaning of the graphical notation we will be using for all the component and connector diagrams produced during the elaboration process.
Step 3: Identify candidate architectural drivers

<table>
<thead>
<tr>
<th>#</th>
<th>Architectural Driver</th>
<th>Importance</th>
<th>Difficulty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Support for new field system</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>2</td>
<td>International language support</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>3</td>
<td>Non-standard unit support</td>
<td>H</td>
<td>M</td>
</tr>
<tr>
<td>4</td>
<td>Latency of event propagation</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>5</td>
<td>Latency of alarm propagation</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>6</td>
<td>Load conditions</td>
<td>H</td>
<td>H</td>
</tr>
</tbody>
</table>

- Architectural drivers 1 through 5 correspond to the quality attribute scenarios (slides 28 and 29)
- Architecture driver 1 is also related to the functionality of the MSLite system (slide 31)
- Architectural driver 6 corresponds to the design constraints (slide 31)
- In addition to the business importance, they are also prioritized for technical difficulty by the architects
Step 4: Choose a design concept that satisfies the architectural drivers

- We start with architecture driver # 1 related to modifiability.
- There are three design concerns related with modifiability:
  - *Localize changes:* this relates to adding a new field system.
  - *Prevention of ripple effects:* this relates to minimizing the number of modules affected as a result of adding a new field system.
  - *Defer binding time:* this relates to the time when a new field system is deployed and the ability of non-programmers to manage such deployment.
- We address these concerns using the *Adaptor* architectural pattern.

<table>
<thead>
<tr>
<th>#</th>
<th>Architectural Driver</th>
<th>Importance</th>
<th>Difficulty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Support for new field system</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>2</td>
<td>International language support</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>3</td>
<td>Non-standard unit support</td>
<td>H</td>
<td>M</td>
</tr>
<tr>
<td>4</td>
<td>Latency of event propagation</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>5</td>
<td>Latency of alarm propagation</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>6</td>
<td>Load conditions</td>
<td>H</td>
<td>H</td>
</tr>
</tbody>
</table>
Introducing adaptors (1)

The *adaptor* architectural pattern by itself has minimal benefit in reducing ripple effects when adding support for a new field system because it does not take into account indirect dependencies.

We use two additional architectural tactics to minimize propagation of change:

- First we specify a standard interface to be exposed by all adaptors (*maintain existing interfaces*).
- Additionally, we use the adaptor as an *intermediary* responsible for semantic translation (when possible).
  - This translation covers for example the unit conversions mentioned in architecture driver # 3.
Introducing adaptors (2)

• Additionally, we use the following two architectural tactics to address the defer binding time design concern:
  – *Runtime registration*: this will support plug-and-play operation allowing non-programmers to deploy new field systems
  – *Configuration files*: this tactic enables setting of configuration parameters (such as initial property values) for the field systems at startup
Step 5: Instantiate architectural elements and allocate responsibilities

- The adaptors are assigned the following responsibilities:
  - Send commands to the FSS
  - Receive events from FSS
  - Perform semantic translation for FSS data

- Instantiating and allocating responsibilities to the adaptors leads to a realization that the MSLite server is still sensitive to a change in the number of field devices it is connected to, and must include logic to route commands and data to and from the correct adaptor

- To address this concern we revisit step 4 and modify our design
Step 4: Choose a design concept that satisfies the architectural drivers

- This is done by introducing a Virtual Field System Simulator (VFSS) pattern
  - The VFSS uses the *hiding information tactic* to hide information about the number and type of field systems actually connected
  - For all other components of the MSLite system, there is practically one field system to interact with at all times
Step 5: Instantiate architectural elements and allocate responsibilities

• The VFSS is assigned the following responsibility:
  – Route data to appropriate FSS
Step 4: Choose a design concept that satisfies the architectural drivers

- We next consider architectural drivers 4, 5 and 6 related to the performance quality attribute.
- There are two design concerns related to these drivers:
  - *Resource Demand*: the arrival of change of property value events from the various field systems and the evaluation of automation rules in response to these events are source of resource demand.
  - *Resource Management*: the demand on resources may have to managed in order to reduce the latency of event and alarm propagation.
- We address these concerns using the *Client-Server* pattern.

<table>
<thead>
<tr>
<th>#</th>
<th>Architectural Driver</th>
<th>Importance</th>
<th>Difficulty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Support for new field system</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>2</td>
<td>International language support</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>3</td>
<td>Non-standard unit support</td>
<td>H</td>
<td>M</td>
</tr>
<tr>
<td>4</td>
<td>Latency of event propagation</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>5</td>
<td>Latency of alarm propagation</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>6</td>
<td>Load conditions</td>
<td>H</td>
<td>H</td>
</tr>
</tbody>
</table>
Introducing clients (1)

- We move the responsibility of rule evaluation and execution, and alarm generation, respectively to a separate Logic & Reaction (L&R) engine component and an Alarm engine component.

- These components running outside the MSLite Server context can now be easily moved to dedicated execution nodes if necessary.
  - In doing so the client-server pattern is making use of the **increase available resources** tactic to address the resource management concern and the **reduce computational overhead** tactic to address the resource demand concern.
Introducing clients (2)

- We use an additional tactic to address the resource management concern:
  - This tactic relies on *introducing concurrency* to reduce delays attributable to “blocked time”
  - The evaluation of automation rules is a prime candidate for concurrency since it is a computationally intensive process with a fairly low and predictable amount of communication
  - Concurrency is used inside the L&R engine to perform simultaneous rule evaluations with the help of thread pools
Step 5: Instantiate architectural elements and allocate responsibilities

- We assign the following responsibilities to the L&R client:
  - Evaluate and execute automation rules
  - Send automation commands
- We assign the following responsibility to the Alarm client:
  - Generate alarm notifications
Step 4: Choose a design concept that satisfies the architectural drivers

- The next to be considered is architecture driver # 3 (non-standard unit support) related to modifiability.
- We have, however, satisfied this driver through the use of an *adaptor* pattern and *intermediary* modifiability tactic (see slide 36).
- We, therefore, move to the final architecture driver # 2 (international language support) also related to modifiability.
- There are two design concerns for this driver:
  - *Localize changes*: this relates to changing the user interface to deal with a new language and culture.
  - *Prevention of ripple effects*: this relates to minimizing the number of modules affected as a result of changing the user interface.
- We address these concerns using the *Model View Controller* architecture pattern.
Separating the user interface

- The user interface (UI) is separated from the rest of the application into a presentation component.
- MVC pattern uses the following modifiability tactics to address localize changes and prevention of ripple effects design concerns:
  - *Anticipation of expected changes:* changes to the UI are localized to the presentation module.
  - *Intermediary:* the controller acts as an intermediary preventing ripple effects from changes to the UI from propagating to the rest of the application.
Step 5: Instantiate architectural elements and allocate responsibilities

- We assign the following responsibilities to the presentation component:
  - Display device data
  - Capture / relay user commands
  - Display alarm conditions
  - Edit / create automation rules
Architecture trade-off

• As previously mentioned, the architecture elaboration process we are using is iterative.

• Moreover, the tactics we choose to implement can very often have a negative impact on the quality attributes they do not target specifically.

• In the case of the MSLite system, we focused on modifiability and performance tactics which can have negative impact on each other.

• We revisit the performance and modifiability drivers next to address these issues.
Step 4: Choose a design concept that satisfies the architectural drivers

- **Revisiting performance drivers**: By examining the current system structure, it can be seen that every time the L&R, the Alarm or the Presentation component needs a value from a field device, it needs to make a call traversing multiple components all the way to the field systems.

- Since crossing component boundaries typically introduces computational overhead, and because the querying latency of field systems is a given constraint over which we have no control, we use the cache architectural pattern to improve device querying performance.
Introducing cache

- This cache provides field device property values to the other system components, saving part of the performance cost incurred when querying the actual field devices. The performance gains are seen because we reduce the number of process and machine boundaries traversed for each query.
  - Cache uses the maintaining multiple copies of data performance tactic
Step 5: Instantiate architectural elements and allocate responsibilities

- We allocate the following responsibilities to the cache:
  - Retrieve object data from FSS
  - Store FSS configuration
Step 4: Choose a design concept that satisfies the architectural drivers

- **Revisiting modifiability drivers**: Introducing performance tactics resulted in creation of multiple components.
- Based on the current structure of the components and type of their connectors, we can predict that some changes to the VFSS have the potential to propagate to five other components (L&R, Alarm, Presentation, Cache and MSLite Server).
- We use the publish subscribe architectural pattern to address this concern.
Introducing publish subscribe bus

- The Publish-Subscribe bus is a component we introduce to implement three modifiability tactics
  - First it alleviates the syntactic dependencies of inter-component calls by acting as a standard interface *intermediary*
  - Second, using the *module generalization* tactic it was made invariant to the type of events it transports
    - This generalization allows new types of events to be transported with no modification to the Publish-Subscribe component
  - Finally, it relies on *runtime registration* to allow system extensibility by adding publishers and subscribers
Step 5: Instantiate architectural elements and allocate responsibilities

- We allocate the following responsibility to the publish subscribe bus:
  - Propagate change of value notifications
The final architecture

- The tactics we apply next are not explicitly stated in the subset of business goals mentioned earlier but are essential non-functional requirements; we apply them here for completeness.
- Security requirements of the system are met by introducing user authentication and user authorization in the access control module.
- “Buildability” is improved by delegating data persistence to an external commercially available database system.
Focus of architecture-centric approach

- The architecture obtained by applying architecture-centric approach reflects its focus on systemic properties the architecture must embody.
- These systemic properties were used as a starting point for creating the architecture.
- This approach does lead to architectures that are more robust when a system’s fitness to purpose is related to systemic or non-functional requirements.
- It does, not, however, address how subsequent design such as the process of identifying component interfaces and their respective operations must occur.
- It should also be noted that in order to generate quality attribute scenarios some understanding of the overall functional requirements of the system is necessary.
  - So, activities similar to OOAD that establish the use case model and business concepts model must also take place.
An integrated approach (1)

- As shown in the previous section, using ADD approach we arrive at an architecture that supports the business and mission goals of the application, but leaves the fine-grained design details unspecified.
- Correspondingly, the traditional OOAD approach arrives at a final design that includes these fine-grained, class-level, details, but these are distributed across an architecture that reflects an emphasis on functional cohesion rather than fundamental business goals.
- Ideally then we would prefer to merge the two approaches to arrive at a final architecture that simultaneously meets business goals and provides sufficient detail for implementation.
An integrated approach (2)

- The following figure presents a process workflow for such an integrated approach.
- It should be noted that this is a partial view putting with emphasis on synergy points between the two methods.
- We voluntarily omit from the figure subsequent activities concerned with constraint specification, provisioning, etc. as they lie beyond the scope of the discussion.
The integrated approach (3)

- Since quality attributes are central to creating an architecture, the architecture derived from architecture-centric methods should form the basis of design and implementation of the system under consideration.
- A feasible approach would be to perform the analysis and domain modeling activities in OOAD, such as the use case and business concepts modeling, that provide a broad understanding of the functional requirements concurrently and iteratively with activities in the architecture-centric approach that provide an understanding of the quality attribute requirements of the system.
The integrated approach (4)

• The quality attribute requirements can then be used for further elaboration of the architecture

• As architectures produced in this manner are high level models, detailed design and implementation of the components and their connectors can be carried out using OOAD

• Such a synergy between the OOAD and architecture-centric approaches provides linkage from high level models to the source code that is important for preserving the integrity of the architectural design as the system evolves (Garlan, 2000)
Detailed design with OOAD (1)

- Before we do the detailed design, it should be noted that the architecture elaboration using architecture-centric approach can be applied recursively to components for creating their internal architecture
- We show this for the Logic and Reaction Engine
- This runtime component and connector view of the Logic and Reaction Engine, shows the Rule Cache, Coordinator, Evaluator, PropertyMapper, Subscription Manager, Command Dispatcher and Event Queue components
  - When events are received, the Coordinator uses the Property Mapper to identify the rules to be evaluated and notifies an evaluator
  - The Evaluator retrieves the rule details from the Rules Cache and communicates the resulting commands to the Command Dispatcher
  - By using concurrency (thread pools) the component is able to achieve different types of performance gains
  - Some of these benefits however are mostly visible when multiple computational nodes are available
Detailed design with OOAD (2)

- The corresponding static structure of the L&R engine is depicted using a design class diagram.
Detailed design with OOAD (3)

- The static structure of the L&R engine is shown in an intermediary state where main methods and attributes are identified and a reduced set of generalizations is applied.
- At this level, it is possible to introduce more design tactics and patterns in iterative refinements.
- This is also the level at which we are able to use results from the OOAD analysis of the domain.
  - To illustrate this connection, the elements from the static structure which have associations with business domain types are shown with a grayed background for differentiation.
Detailed design with OOAD (4)

- The reference OOAD methodology we used offers a systematic process for identifying system and business interfaces and discovering their respective operations.

- At this specification level, a significant portion of these interfaces is independent of the architectural decisions since it derives mainly from the domain and requirement analysis.

- In our integrated approach, we rely on these interfaces for specifying the responsibilities of components and connectors.
Conclusions (1)

- Every system has a rationale for its creation which takes the form of business goals set forth by the organization creating the system and has a strong influence on the architecture of the system under consideration.
- There is little systematic guidance to software architects in existing software development methodologies, such as OOAD, on how to create architectures that support the business goals of an organization.
  - In OOAD, architectures simply emerge from the design and architects are left to rely on their skill level and experience to ensure this emergent architecture converges on a suitable architecture that reflects the business goals.
- OOAD strives for semantic closeness to the domain and functional cohesion and primarily uses functional decomposition guided by principles of abstraction, encapsulation, information hiding and separation of concerns to define the structure of the system.
- Architecture centric methods use architectural patterns and tactics associated with systemic properties as a guide for decomposing a system.
Conclusions (2)

• Architectural patterns and tactics codify design decisions and layout an architectural strategy for achieving a quality attribute goal (Bass et al., 2003)
  – They provide constraints, akin to load-bearing walls in a building, that clearly prescribe what must be preserved to prevent architectural erosion and drift as the system evolves (Perry and Wolf, 1992)
  – An architecture based on these tactics offers an improved opportunity for analyzing a system and its ability to satisfy its quality attribute requirements (Bass et al., 2003; Shaw and Garlan, 1996; Perry and Wolf, 1992)

• In this talk, we analyzed a system called MSLite from the building automation domain to demonstrate the differences between OOAD and architecture-centric approaches, and investigated the feasibility of an integrated approach that provides the benefits of both

• Using architecture-centric methods we were clearly able to take into account the business goals and their related quality attributes in formulating a high level architecture

• The analysis activities of OOAD provided a broad functional understanding of the system that served as input to the quality attribute scenarios used for the high level architecture

• This architecture was then used as a basis for doing further detailed design and implementation using OOAD.
References (1)

References (2)