SENG 401
Software Architecture

Lecture 3
Topic: Documenting Software Architecture
Sub-Topic: Module Views

Gias Uddin
Assistant Professor, University of Calgary

http://giasuddin.ca/

Books Used/Cited in this Course

ID	Title
B1	Software Architecture: Foundations, Theory, and Practice. Taylor, Medvidovic, Dashofy. Wiley, 2009. *Available at UofC Library*
B2	The Architecture of Open-Source Applications. Amy Brown, Greg Wilson. Volume I. http://www.aosabook.org/en/index.html
B3	The Architecture of Open-Source Applications. Amy Brown, Greg Wilson. Volume II. http://www.aosabook.org/en/index.html
B4	**Documenting Software Architecture: Views and Beyond, 2nd Edition. Addison-Wesley, 2010. Clements et al.** ***Available at UofC Library***
B5	Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives, 2nd Edition. Nick Rozanski. Addison-Wesley Professions, 2011. *Available at UofC Library*
B6	Software Architecture in Practice. Bass, Clements, Kazman. Addison-Wesley, 2012 *Available at UofC Library*

SCHULICH
School of Engineering

Topic 3
Documenting Software Architecture
Collection of Software Architectural Styles
Part 1. Module Views

References: B4 – C1

Why document software architecture

SCHULICH School of Engineering

Some of the stakeholders of architecture documentation, their roles, and how they might use it

Stakeholder	How
Analyst	Analyzes system architecture to ensure it meets certain critical quality attributes (performance, safety, security, etc.).
Architect	Responsible for the design and documentation of architecture. Negotiates and makes trade-offs among competing requirements and design approaches. Provides evidence that the architecture satisfies the requirements.
Business Manager	Responsible for the functioning of the business/organizational entity that owns the system. Understands the ability of the architecture to meet business goals.
Customer	Pays for the system and ensures its delivery. Assures required functionality and quality will be delivered.
Project Manager	Responsible for planning, scheduling and allocating resources. Helps to set budget and schedule and to identify and resolve resource contention.
Product Line Manager	Responsible for the development and leading of an entire family or products following the system architecture.
Tester	Responsible for the test and verification of system and its elements against the formal requirement and architecture.
Many more ….	

Module Views

Overview

- Module views enumerate the principal implementation units, or modules, of a system, together with the relations among these units.
- The way in which a system's software is decomposed into manageable units remains one of the important forms of system structure. At a minimum, it determines how a system's source code is decomposed into units, what kinds of assumptions each unit can make about services provided by other units, and how those units are aggregated into larger ensembles. It also includes global data structures that impact and are impacted by multiple units. Module structures often determine how changes to one part of a system might affect other parts and hence the ability of a system to support modifiability, portability, and reuse.
- It is unlikely that the documentation of any software architecture can be complete without at least one module view.

Module Views
What It's For

Task	Description
Construction	A module view can provide a blueprint for the source code and the data store. In this case, the modules and physical structures, such as source code files and directories, often have a close mapping.
Analysis	Two important analysis techniques are requirements traceability and impact analysis. Because modules partition the system, it should be possible to determine how the functional requirements of a system are supported by module responsibilities. Some functional requirements will be met by a sequence of invocations among modules. Documenting such sequences shows how the system is meeting its requirements and identifies any unaddressed requirements.
Communication	The various levels of granularity of the module decomposition provide a top-down presentation of the system's responsibilities and therefore can guide the learning process. For a system whose implementation is already in place, module views, if kept up to date, are very helpful, as they explain the structure of the code base to a new developer on the team—much more effective than providing the URL to the version management system repository and asking him or her to browse the source files and read the code. Thus, up-to-date module views are very useful during system maintenance.

Module Views
Key Terms

SCHULICH School of Engineering

Term	Definition
Elements	Modules, which are implementation units of software that provide a coherent set of responsibilities.
Relations	• ***Is Part of*** : defines a part/whole relationship between the submodule – the part – and the aggregate module – the whole. • ***Depends on*** : defines the dependency relationship between two modules. Specific module styles elaborate what dependency is meant. • ***Is a*** : defines a generalization/specialization relationship between a more specific module – the child – and a more general module – the parent.
Properties	Different module views may impose specific topological properties & constraints

Module Terms
Elements

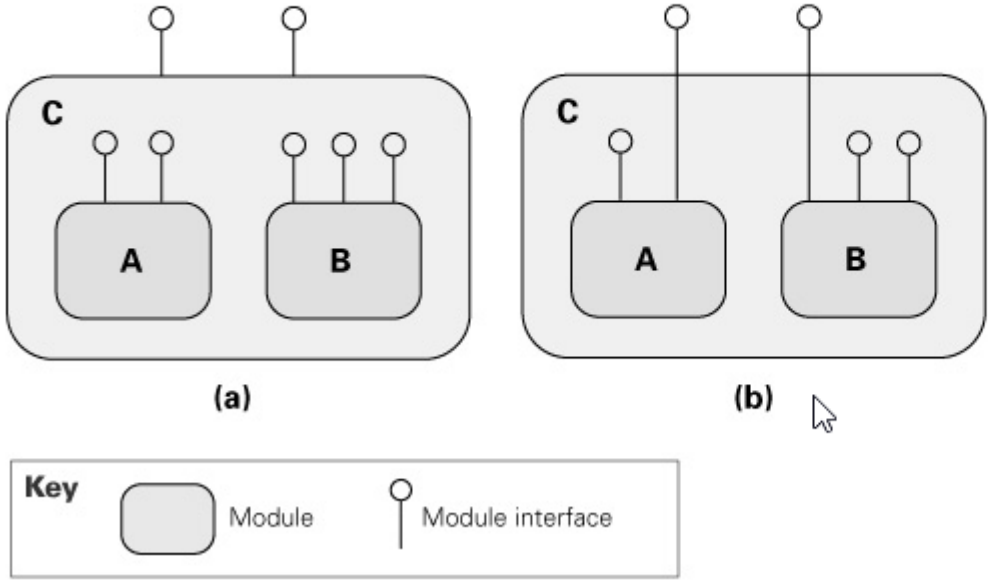
- System designers use the term module to refer to a wide variety of software structures, including programming language units—such as C programs, Java or C# classes, Delphi units, and PL/SQL stored procedures—or simply general groupings of source code units—such as Java packages or C# namespaces.
- We characterize a module by enumerating its set of responsibilities, which are foremost among a module's properties. This broad notion of responsibilities is meant to encompass the kinds of features that a unit of software might provide: that is, its functionality and the knowledge it maintains.
- Modules can be aggregated and decomposed. Each of the various module styles identifies a different set of modules and relations, and then aggregates or decomposes these modules based on relevant style criteria. For example, the layered style identifies modules and aggregates them based on an allowed-to-use relation, whereas the generalization style identifies and aggregates modules based on what they have in common.

Module Terms
Properties

Property	Description
Name	A module's name is the primary means to refer it. A module's name often suggests something about its role in the system: a module called "account_mgr," for instance, probably has little to do with numeric simulations of chemical reactions. In addition, a module's name may reflect its position in a decomposition hierarchy; the name "A.B.C," for example, refers to a module C that is a submodule of a module B, itself a submodule of A.
Responsibility	The responsibility property of a module is a way to identify its role in the overall system and establishes an identity for it beyond the name. Whereas a module's name may suggest its role, a statement of responsibility establishes it with much more certainty. Responsibilities should be described in sufficient detail to make clear to the reader what each module does.
Visibility of interfaces	When a module has submodules, some interfaces of the submodules may have internal purposes; that is, the interfaces are used only by the submodules within the enclosing parent module. These interfaces are not visible outside that context and therefore do not have a direct relationship to the parent interfaces.

Module Terms - Properties
Visibility of interfaces - Encapsulation

Figure 1.1 (a) Module C provides its own interface, hiding the interfaces of modules A and B. (b) Module C exposes a subset of the interfaces of modules A and B as its interface.



Module Terms
Properties - continued

Property	Description
Implementation Information	Because modules are units of implementation, it is useful to record information related to their implementation from the point of view of managing their development and building the system that contains them. Although this information is not, strictly speaking, architectural, it may be useful to record it in the architecture documentation where the module is defined. Implementation information might include
Mapping to source code unit	This identifies the files that constitute the implementation of a module. For example, a module named Account, if implemented in Java, might have several files that constitute its implementation: IAccount.java (an interface), AccountImpl.java (an implementation of Account functionality), AccountBean.java (a class to hold the state of an account in memory), AccountOrmMapping.xml (a file that defines the mapping between AccountBean and a database table—object-relational mapping), and perhaps even a unit test AccountTest.java.

Module Terms
Properties - continued

Property	Description
Test information	The module's test plan, test cases, test scaffolding, and test data are important to store.
Management information	A manager may need information about the module's predicted schedule and budget.
Implementation constraints	In many cases, the architect will have a certain implementation strategy in mind for a module or may know of constraints that the implementation must follow. This information is private to the module and hence will not appear, for example, in the module's interface.

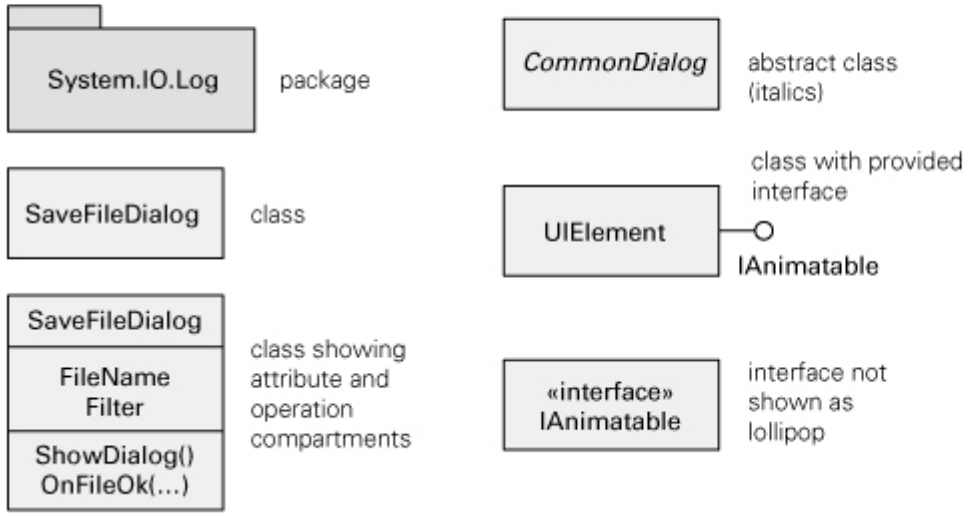
Module Views Notations Used in this Course

SCHULICH School of Engineering

Property	Description
UML	UML is unified modeling language. Software modeling notations, such as UML, provide a variety of constructs that can be used to represent modules. UML has a class construct, which is the object-oriented specialization of a module as described here. UML packages are used to represent an aggregation of modules. UML packages can represent, for example, layers, subsystems, and collections of implementation units that live together in the implementation namespace. UML was originally created to model object-oriented systems. It is now considered a general-purpose modeling language. As a result, UML elements and relations are generic; that is, they are not specific to implementation technologies or platforms.
ER Diagram	An entity-relationship diagram (ERD) is a notation specifically used for data modeling. It shows data entities that require a representation in the system and their relationships. These relationships can be one-to-one, one-to-many, or many-to-many.

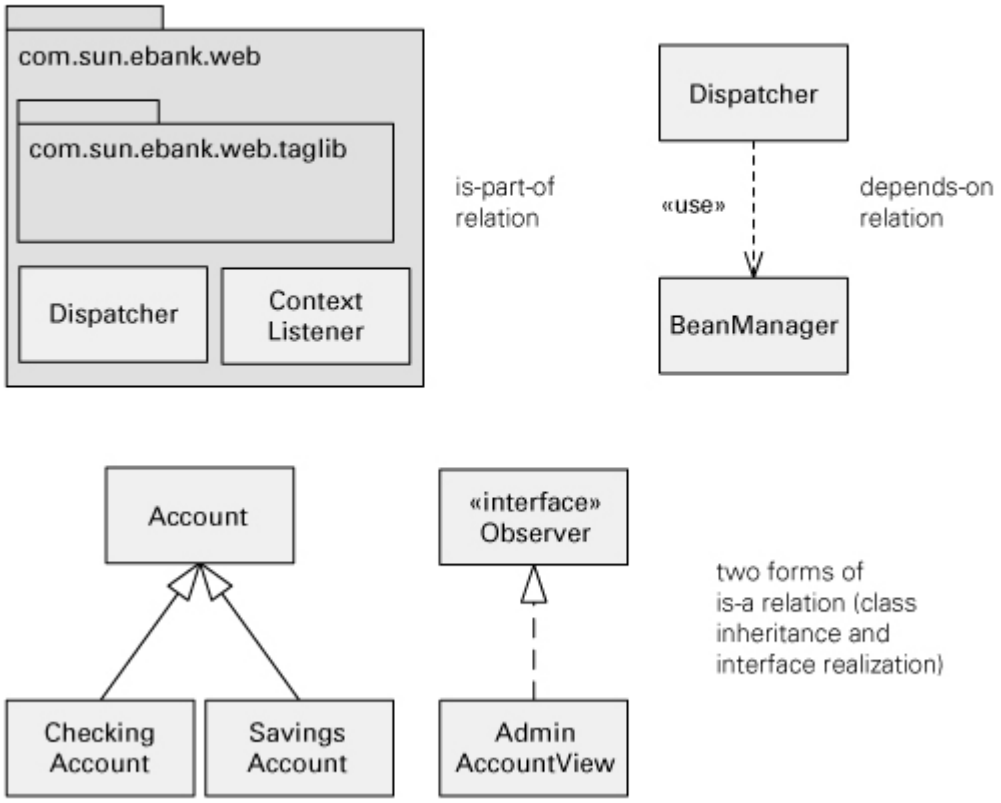
Module Views
UML Notations

Figure 1.2 Examples of module notation in UML. A module may be represented as a class or a package. More specific types of modules can be indicated with stereotypes (as in Figure 1.4).



Module Views
UML Notations

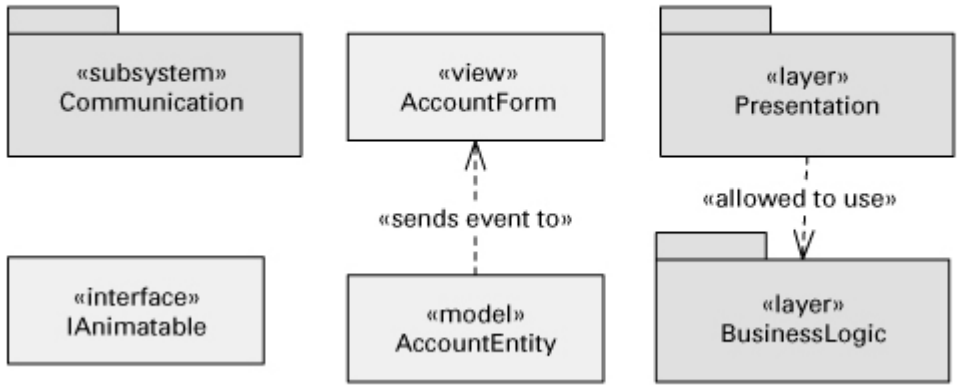
Figure 1.3 Examples of module relations in UML



Module Views Notations
Stereotypes in UML

Stereotype is a UML extension mechanism that allows the definition of a new type of modeling element or relation based on an existing UML element or relation. A **stereotype** is a UML extension mechanism and is represented in diagrams as a label in guillemets («stereotype label»). Figure 1.4 shows some examples. If used correctly, stereotypes make your UML diagrams more expressive. The UML specification provides a number of standard stereotypes, but you can also create your own.

Figure 1.4 Examples of UML elements and relations with stereotypes



Acknowledgment

Lecture contents are taken from book B4 Chapter C1.