UML Summary 1

The UML Summary provides an introduction to the UML, discussing its motivation and history.

Contents

This chapter contains the following topics.

1.1 Overview

The Unified Modeling Language (UML) is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

1.2 Primary Artifacts of the UML

What are the primary artifacts of the UML? This can be answered from two different perspectives: the UML definition itself and how it is used to produce project artifacts.

1.2.1 UML-defining Artifacts

To aid the understanding of the artifacts that constitute the Unified Modeling Language itself, this document consists of the UML Semantics, UML Notation Guide, and UML Extensions sections.

1.2.2 Development Project Artifacts

The choice of what models and diagrams one creates has a profound influence upon how a problem is attacked and how a corresponding solution is shaped. Abstraction, the focus on relevant details while ignoring others, is a key to learning and communicating. Because of this:

- Every complex system is best approached through a small set of nearly independent views of a model. No single view is sufficient.
- Every model may be expressed at different levels of fidelity.
- The best models are connected to reality.

In terms of the views of a model, the UML defines the following graphical diagrams:

- use case diagram
- class diagram
- behavior diagrams:
  - statechart diagram
  - activity diagram
  - interaction diagrams:
    - sequence diagram
    - collaboration diagram
- implementation diagrams:
  - component diagram
  - deployment diagram

Although other names are sometimes given to these diagrams, this list constitutes the canonical diagram names.

These diagrams provide multiple perspectives of the system under analysis or development. The underlying model integrates these perspectives so that a self-consistent system can be analyzed and built. These diagrams, along with supporting documentation, are the primary artifacts that a modeler sees, although the UML and supporting tools will provide for a number of derivative views. These diagrams are further described in the UML Notation Guide (Section 3).

A frequently asked question has been, "Why doesn't UML support data-flow diagrams?" Simply put, data-flow and other diagram types that were not included in the UML do not fit as cleanly into a consistent object-oriented paradigm. Activity diagrams accomplish much of what people want from DFDs, and then some. Activity diagrams are also useful for modeling workflow.

1.3 Motivation to Define the UML

This section describes several factors motivating the UML and includes why modeling is essential, it highlights a few key trends in the software industry, and describes the issues caused by divergence of modeling approaches.

1.3.1 Why We Model

Developing a model for an industrial-strength software system prior to its construction or renovation is as essential as having a blueprint for large building. Good models are essential for communication among project teams and to assure architectural soundness. We build models of complex systems because we cannot comprehend any such system in its entirety. As the complexity of systems increase, so does the importance of good modeling techniques. There are many additional factors of a project's success, but having a rigorous modeling language standard is one essential factor. A modeling language must include:

- Model elements — fundamental modeling concepts and semantics
- Notation — visual rendering of model elements
- Guidelines — idioms of usage within the trade

In the face of increasingly complex systems, visualization and modeling become essential. The UML is a well-defined and widely accepted response to that need. It is the visual modeling language of choice for building object-oriented and component-based systems.

1.3.2 Industry Trends in Software

As the strategic value of software increases for many companies, the industry looks for techniques to automate the production of software. We look for techniques to improve quality and reduce cost and time-to-market. These techniques include component technology, visual programming, patterns, and frameworks. We also seek techniques to manage the complexity of systems as they increase in scope and scale. In particular, we recognize the need to solve recurring architectural problems, such as physical distribution, concurrency, replication, security, load balancing, and fault tolerance. Development for the worldwide web makes some things simpler, but exacerbates these architectural problems.

Complexity will vary by application domain and process phase. One of the key motivations in the minds of the UML developers was to create a set of semantics and notation that adequately addresses all scales of architectural complexity, across all domains.

1.3.3 Prior to Industry Convergence

Prior to the UML, there was no clear leading modeling language. Users had to choose from among many similar modeling languages with minor difference in overall expressive power. Most of the modeling languages shared a set of commonly accepted concepts that are expressed slightly differently in various languages. This lack of agreement discouraged new users from entering the OO market and from doing OO modeling, without greatly expanding the power of modeling. Users longed for the industry to adopt one, or a very few, broadly supported modeling languages suitable for general-purpose usage.

Some vendors were discouraged from entering the OO modeling area because of the need to support many similar, but slightly different, modeling languages. In particular, the supply of add-on tools has been depressed because small vendors cannot afford to support many different formats from many different front-end modeling tools. It is important to the entire OO industry to encourage broadly based tools and vendors, as well as niche products that cater to the needs of specialized groups.

The perpetual cost of using and supporting many modeling languages motivated many companies producing or using OO technology to endorse and support the development of the UML.

While the UML does not guarantee project success, it does improve many things. For example, it significantly lowers the perpetual cost of training and retooling when changing between projects or organizations. It provides the opportunity for new integration between tools, processes, and domains. But most importantly, it enables developers to focus on delivering business value and gives them a paradigm to accomplish this.

1.4 Goals of the UML

The primary design goals of the UML are as follows:

- Provide users with a ready-to-use, expressive visual modeling language to develop and exchange meaningful models.
- Provide extensibility and specialization mechanisms to extend the core concepts.
- Be independent of particular programming languages and development processes.
- Provide a formal basis for understanding the modeling language.
- Encourage the growth of the OO tools market.
- Support higher-level development concepts such as collaborations, frameworks, patterns, and components.
- Integrate best practices.

These goals are discussed in detail below.

Provide users with a ready-to-use, expressive visual modeling language to develop and exchange meaningful models

It is important that the OOAD standard supports a modeling language that can be used "out of the box" to do normal general-purpose modeling tasks. If the standard merely provides a meta-meta-description that requires tailoring to a particular set of modeling concepts, then it will not achieve the purpose of allowing users to exchange models without losing information or without imposing excessive work to map their models to a very abstract form. The UML consolidates a set of core modeling concepts that are generally accepted across many current methods and modeling tools. These concepts are needed in many or most large applications, although not every concept is needed in every part of every application. Specifying a meta-meta-level format for the concepts is not sufficient for model users, because the concepts must be made concrete for real modeling to occur. If the concepts in different application areas were substantially different, then such an approach might work, but the core concepts needed by most application areas are similar and should be supported directly by the standard without the need for another layer.

Provide extensibility and specialization mechanisms to extend the core concepts

OMG expects that the UML will be tailored as new needs are discovered and for specific domains. At the same time, we do not want to force the common core concepts to be redefined or re-implemented for each tailored area. Therefore, we believe that the extension mechanisms should support deviations from the common case, rather than being required to implement the core OOA&D concepts themselves. The core concepts should not be changed more than necessary. Users need to be able to

- build models using core concepts without using extension mechanisms for most normal applications,
- add new concepts and notations for issues not covered by the core,
- choose among variant interpretations of existing concepts, when there is no clear consensus, and
- specialize the concepts, notations, and constraints for particular application domains.

Be independent of particular programming languages and development processes

The UML must and can support all reasonable programming languages. It also must and can support various methods and processes of building models. The UML can support multiple programming languages and development methods without excessive difficulty.

Provide a formal basis for understanding the modeling language

Because users will use formality to help understand the language, it must be both precise and approachable; a lack of either dimension damages its usefulness. The formalisms must not require excessive levels of indirection or layering, use of low-level mathematical notations distant from the modeling domain, such as set-theoretic notation, or operational definitions that are equivalent to programming an

implementation. The UML provides a formal definition of the static format of the model using a metamodel expressed in UML class diagrams. This is a popular and widely accepted formal approach for specifying the format of a model and directly leads to the implementation of interchange formats. UML expresses well-formedness constraints in precise natural language plus Object Constraint Language expressions. UML expresses the operational meaning of most constructs in precise natural language. The fully formal approach taken to specify languages such as Algol-68 was not approachable enough for most practical usage.

Encourage the growth of the OO tools market

By enabling vendors to support a standard modeling language used by most users and tools, the industry benefits. While vendors still can add value in their tool implementations, enabling interoperability is essential. Interoperability requires that models can be exchanged among users and tools without loss of information. This can only occur if the tools agree on the format and meaning of all of the relevant concepts. Using a higher meta-level is no solution unless the mapping to the user-level concepts is included in the standard.

Support higher-level development concepts such as collaborations, frameworks, patterns, and components

Clearly defined semantics of these concepts is essential to reap the full benefit of OO and reuse. Defining these within the holistic context of a modeling language is a unique contribution of the UML.

Integrate best practices

A key motivation behind the development of the UML has been to integrate the best practices in the industry, encompassing widely varying views based on levels of abstraction, domains, architectures, life cycle stages, implementation technologies, etc. The UML is indeed such an integration of best practices.

1.5 Scope of the UML

The Unified Modeling Language (UML) is a language for specifying, constructing, visualizing, and documenting the artifacts of a software-intensive system.

First and foremost, the Unified Modeling Language fuses the concepts of Booch, OMT, and OOSE. The result is a single, common, and widely usable modeling language for users of these and other methods.

Second, the Unified Modeling Language pushes the envelope of what can be done with existing methods. As an example, the UML authors targeted the modeling of concurrent, distributed systems to assure the UML adequately addresses these domains.

Third, the Unified Modeling Language focuses on a standard modeling language, not a standard process. Although the UML must be applied in the context of a process, it is our experience that different organizations and problem domains require different processes. (For example, the development process for shrink-wrapped software is an

interesting one, but building shrink-wrapped software is vastly different from building hard-real-time avionics systems upon which lives depend.) Therefore, the efforts concentrated first on a common metamodel (which unifies semantics) and second on a common notation (which provides a human rendering of these semantics). The UML authors promote a development process that is use-case driven, architecture centric, and iterative and incremental.

The UML specifies a modeling language that incorporates the object-oriented community's consensus on core modeling concepts. It allows deviations to be expressed in terms of its extension mechanisms. The Unified Modeling Language provides the following:

- Sufficient semantics and notation to address a wide variety of contemporary modeling issues in a direct and economical fashion.
- Sufficient semantics to address certain expected future modeling issues, specifically related to component technology, distributed computing, frameworks, and executability.
- Extensibility mechanisms so individual projects can extend the metamodel for their application at low cost. We don't want users to adjust the UML metamodel itself.
- Extensibility mechanisms so that future modeling approaches could be grown on top of the UML.
- Sufficient semantics to facilitate model interchange among a variety of tools.
- Sufficient semantics to specify the interface to repositories for the sharing and storage of model artifacts.

1.5.1 Outside the Scope of the UML

Programming Languages

The UML, a visual modeling language, is not intended to be a visual programming language, in the sense of having all the necessary visual and semantic support to replace programming languages. The UML is a language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system, but it does draw the line as you move toward code. For example, complex branches and joins are better expressed in a textual programming language. The UML does have a tight mapping to a family of OO languages so that you can get the best of both worlds.

Tools

Standardizing a language is necessarily the foundation for tools and process. Tools and their interoperability are very dependent on a solid semantic and notation definition, such as the UML provides. The UML defines a semantic metamodel, not a tool interface, storage, or run-time model, although these should be fairly close to one another.

The UML documents do include some tips to tool vendors on implementation choices, but do not address everything needed. For example, they don't address topics like diagram coloring, user navigation, animation, storage/implementation models, or other features.

Process

Many organizations will use the UML as a common language for its project artifacts, but will use the same UML diagram types in the context of different processes. The UML is intentionally process independent, and defining a standard process was not a goal of the UML or OMG's RFP.

The UML authors do recognize the importance of process. The presence of a well-defined and well-managed process is often a key discriminator between hyperproductive projects and unsuccessful ones. The reliance upon heroic programming is not a sustainable business practice. A process

- provides guidance as to the order of a team's activities,
- specifies what artifacts should be developed,
- directs the tasks of individual developers and the team as a whole, and
- offers criteria for monitoring and measuring a project's products and activities.

Processes by their very nature must be tailored to the organization, culture, and problem domain at hand. What works in one context (shrink-wrapped software development, for example) would be a disaster in another (hard-real-time, human-rated systems, for example). The selection of a particular process will vary greatly, depending on such things as problem domain, implementation technology, and skills of the team.

Booch, OMT, OOSE, and many other methods have well-defined processes, and the UML can support most methods. There has been some convergence on development process practices, but there is not yet consensus for standardization. What will likely result is general agreement on best practices and potentially the embracing of a process framework, within which individual processes can be instantiated. Although the UML does not mandate a process, its developers have recognized the value of a use-case driven, architecture-centric, iterative, and incremental process, so were careful to enable (but not require) this with the UML.

1.5.2 Comparing UML to Other Modeling Languages

It should be made clear that the Unified Modeling Language is not a radical departure from Booch, OMT, or OOSE, but rather the legitimate successor to all three. This means that if you are a Booch, OMT, or OOSE user today, your training, experience, and tools will be preserved, because the Unified Modeling Language is a natural evolutionary step. The UML will be equally easy to adopt for users of many other methods, but their authors must decide for themselves whether to embrace the UML concepts and notation underneath their methods.

The Unified Modeling Language is more expressive yet cleaner and more uniform than Booch, OMT, OOSE, and other methods. This means that there is value in moving to the Unified Modeling Language, because it will allow projects to model things they could not have done before. Users of most other methods and modeling languages will gain value by moving to the UML, since it removes the unnecessary differences in notation and terminology that obscure the underlying similarities of most of these approaches.

With respect to other visual modeling languages, including entity-relationship modeling, BPR flow charts, and state-driven languages, the UML should provide improved expressiveness and holistic integrity.

Users of existing methods will experience slight changes in notation, but this should not take much relearning and will bring a clarification of the underlying semantics. If the unification goals have been achieved, UML will be an obvious choice when beginning new projects, especially as the availability of tools, books, and training becomes widespread. Many visual modeling tools support existing notations, such as Booch, OMT, OOSE, or others, as views of an underlying model; when these tools add support for UML (as some already have) users will enjoy the benefit of switching their current models to the UML notation without loss of information.

Existing users of any OO method can expect a fairly quick learning curve to achieve the same expressiveness as they previously knew. One can quickly learn and use the basics productively. More advanced techniques, such as the use of stereotypes and properties, will require some study since they enable very expressive and precise models needed only when the problem at hand requires them.

1.5.3 Features of the UML

The goals of the unification efforts were to keep it simple, to cast away elements of existing Booch, OMT, and OOSE that didn't work in practice, to add elements from other methods that were more effective, and to invent new only when an existing solution was not available. Because the UML authors were in effect designing a language (albeit a graphical one), they had to strike a proper balance between minimalism (everything is text and boxes) and over-engineering (having an icon for every conceivable modeling element). To that end, they were very careful about adding new things, because they didn't want to make the UML unnecessarily complex. Along the way, however, some things were found that were advantageous to add because they have proven useful in practice in other modeling.

There are several new concepts that are included in UML, including

- extensibility mechanisms (stereotypes, tagged values, and constraints),
- threads and processes,
- distribution and concurrency (e.g., for modeling ActiveX/DCOM and CORBA),
- patterns/collaborations,
- activity diagrams (for business process modeling),
- refinement (to handle relationships between levels of abstraction),

- interfaces and components, and
- a constraint language.

Many of these ideas were present in various individual methods and theories but UML brings them together into a coherent whole. In addition to these major changes, there are many other localized improvements over the Booch, OMT, and OOSE semantics and notation.

The UML is an evolution from Booch, OMT, OOSE, other object-oriented methods, and many other sources. These various sources incorporated many different elements from many authors, including non-OO influences. The UML notation is a melding of graphical syntax from various sources, with a number of symbols removed (because they were confusing, superfluous, or little used) and with a few new symbols added. The ideas in the UML come from the community of ideas developed by many different people in the object-oriented field. The UML developers did not invent most of these ideas; rather, their role was to select and integrate the best ideas from OO and computer-science practices. The actual genealogy of the notation and underlying detailed semantics is complicated, so it is discussed here only to provide context, not to represent precise history.

Use-case diagrams are similar in appearance to those in OOSE.

Class diagrams are a melding of OMT, Booch, class diagrams of most other OO methods. Extensions (e.g., stereotypes and their corresponding icons) can be defined for various diagrams to support other modeling styles. Stereotypes, constraints, and taggedValues are concepts added in UML that did not previously exist in the major modeling languages.

Statechart diagrams are substantially based on the statecharts of David Harel with minor modifications. The Activity diagram, which shares much of the same underlying semantics, is similar to the work flow diagrams developed by many sources including many pre-OO sources.

Sequence diagrams were found in a variety of OO methods under a variety of names (interaction, message trace, and event trace) and date to pre-OO days. Collaboration diagrams were adapted from Booch (object diagram), Fusion (object interaction graph), and a number of other sources.

Collaborations are now first-class modeling entities, and often form the basis of patterns.

The implementation diagrams (component and deployment diagrams) are derived from Booch's module and process diagrams, but they are now component-centered, rather than module-centered and are far better interconnected.

Stereotypes are one of the extension mechanisms and extend the semantics of the metamodel. User-defined icons can be associated with given stereotypes for tailoring the UML to specific processes.

Object Constraint Language is used by UML to specify the semantics and is provided as a language for expressions during modeling. OCL is an expression language having its root in the Syntropy method and has been influenced by expression languages in other methods like Catalysis. The informal navigation from OMT has the same intent, where OCL is formalized and more extensive.

Each of these concepts has further predecessors and many other influences. We realize that any brief list of influences is incomplete and we recognize that the UML is the product of a long history of ideas in the computer science and software engineering area.

1.6 UML - Past, Present, and Future

The UML was developed by Rational Software and its partners. Many companies are incorporating the UML as a standard into their development process and products, which cover disciplines such as business modeling, requirements management, analysis & design, programming, and testing.

1.6.1 UML 0.8 - 0.91

Precursors to UML

Identifiable object-oriented modeling languages began to appear between mid-1970 and the late 1980s as various methodologists experimented with different approaches to object-oriented analysis and design. Several other techniques influenced these languages, including Entity-Relationship modeling, the Specification & Description Language (SDL, circa 1976, CCITT), and other techniques. The number of identified modeling languages increased from less than 10 to more than 50 during the period between 1989-1994. Many users of OO methods had trouble finding complete satisfaction in any one modeling language, fueling the "method wars." By the mid-1990s, new iterations of these methods began to appear, most notably Booch ’93, the continued evolution of OMT, and Fusion. These methods began to incorporate each other’s techniques, and a few clearly prominent methods emerged, including the OOSE, OMT-2, and Booch ’93 methods. Each of these was a complete method, and was recognized as having certain strengths. In simple terms, OOSE was a use-case oriented approach that provided excellent support business engineering and requirements analysis. OMT-2 was especially expressive for analysis and data-intensive information systems. Booch ’93 was particularly expressive during design and construction phases of projects and popular for engineering-intensive applications.

Booch, Rumbaugh, and Jacobson Join Forces

The development of UML began in October of 1994 when Grady Booch and Jim Rumbaugh of Rational Software Corporation began their work on unifying the Booch and OMT (Object Modeling Technique) methods. Given that the Booch and OMT methods were already independently growing together and were collectively recognized as leading object-oriented methods worldwide, Booch and Rumbaugh joined forces to forge a complete unification of their work. A draft version 0.8 of the

Unified Method, as it was then called, was released in October of 1995. In the Fall of 1995, Ivar Jacobson and his Objectory company joined Rational and this unification effort, merging in the OOSE (Object-Oriented Software Engineering) method. The Objectory name is now used within Rational primarily to describe its UML-compliant process, the Rational Objectory Process.

As the primary authors of the Booch, OMT, and OOSE methods, Grady Booch, Jim Rumbaugh, and Ivar Jacobson were motivated to create a unified modeling language for three reasons. First, these methods were already evolving toward each other independently. It made sense to continue that evolution together rather than apart, eliminating the potential for any unnecessary and gratuitous differences that would further confuse users. Second, by unifying the semantics and notation, they could bring some stability to the object-oriented marketplace, allowing projects to settle on one mature modeling language and letting tool builders focus on delivering more useful features. Third, they expected that their collaboration would yield improvements in all three earlier methods, helping them to capture lessons learned and to address problems that none of their methods previously handled well.

As they began their unification, they established four goals to focus their efforts:

1. Enable the modeling of systems (and not just software) using object-oriented concepts
2. Establish an explicit coupling to conceptual as well as executable artifacts
3. Address the issues of scale inherent in complex, mission-critical systems
4. Create a modeling language usable by both humans and machines

Devising a notation for use in object-oriented analysis and design is not unlike designing a programming language. There are tradeoffs. First, one must bound the problem: Should the notation encompass requirement specification? (Yes, partially.) Should the notation extend to the level of a visual programming language? (No.) Second, one must strike a balance between expressiveness and simplicity: Too simple a notation will limit the breadth of problems that can be solved; too complex a notation will overwhelm the mortal developer. In the case of unifying existing methods, one must also be sensitive to the installed base: Make too many changes, and you will confuse existing users. Resist advancing the notation, and you will miss the opportunity of engaging a much broader set of users. The UML definition strives to make the best tradeoffs in each of these areas.

The efforts of Booch, Rumbaugh, and Jacobson resulted in the release of the UML 0.9 and 0.91 documents in June and October of 1996. During 1996, the UML authors invited and received feedback from the general community. They incorporated this feedback, but it was clear that additional focused attention was still required.

1.6.2 UML Partners

During 1996, it became clear that several organizations saw UML as strategic to their business. A Request for Proposal (RFP) issued by the Object Management Group (OMG) provided the catalyst for these organizations to join forces around producing a

joint RFP response. Rational established the UML Partners consortium with several organizations willing to dedicate resources to work toward a strong UML definition. Those contributing most to the UML definition included: Digital Equipment Corp., HP, i-Logix, IntelliCorp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle, Rational Software, TI, and Unisys. This collaboration produced UML, a modeling language that was well defined, expressive, powerful, and generally applicable.

In January 1997 IBM & ObjecTime; Platinum Technology; Ptech; Taskon & Reich Technologies; and Softeam also submitted separate RFP responses to the OMG. These companies joined the UML partners to contribute their ideas, and together the partners produced the revised UML 1.1 response. The focus of the UML 1.1 release was to improve the clarity of the UML 1.0 semantics and to incorporate contributions from the new partners.

This document is based on the UML 1.1 release and is the result of a collaborative team effort. The UML Partners have worked hard as a team to define UML. While each partner came in with their own perspective and areas of interest, the result has benefited from each of them and from the diversity of their experiences. The UML Partners contributed a variety of expert perspectives, including, but not limited to, the following: OMG and RM-ODP technology perspectives, business modeling, constraint language, state machine semantics, types, interfaces, components, collaborations, refinement, frameworks, distribution, and metamodel.

1.6.3 UML - Present and Future

The UML is nonproprietary and open to all. It addresses the needs of user and scientific communities, as established by experience with the underlying methods on which it is based. Many methodologists, organizations, and tool vendors have committed to use it. Since the UML builds upon similar semantics and notation from Booch, OMT, OOSE, and other leading methods and has incorporated input from the UML partners and feedback from the general public, widespread adoption of the UML should be straightforward.

There are two aspects of "unified" that the UML achieves: First, it effectively ends many of the differences, often inconsequential, between the modeling languages of previous methods. Secondly, and perhaps more importantly, it unifies the perspectives among many different kinds of systems (business versus software), development phases (requirements analysis, design, and implementation), and internal concepts.

Standardization of the UML

Many organizations have already endorsed the UML as their organization’s standard, since it is based on the modeling languages of leading OO methods. The UML is ready for widespread use. This document is suitable as the primary source for authors writing books and training materials, as well as developers implementing visual modeling tools. Additional collateral, such as articles, training courses, examples, and books, will soon make the UML very approachable for a wide audience.

Industrialization

Many organizations and vendors worldwide have already embraced the UML. The number of endorsing organizations is expected to grow significantly over time. These organizations will continue to encourage the use of the Unified Modeling Language by making the definition readily available and by encouraging other methodologists, tool vendors, training organizations, and authors to adopt the UML.

The real measure of the UML's success is its use on successful projects and the increasing demand for supporting tools, books, training, and mentoring.

Future UML Evolution

Although the UML defines a precise language, it is not a barrier to future improvements in modeling concepts. We have addressed many leading-edge techniques, but expect additional techniques to influence future versions of the UML. Many advanced techniques can be defined using UML as a base. The UML can be extended without redefining the UML core.

The UML, in its current form, is expected to be the basis for many tools, including those for visual modeling, simulation, and development environments. As interesting tool integrations are developed, implementation standards based on the UML will become increasingly available.

The UML has integrated many disparate ideas, so this integration will accelerate the use of OO. Component-based development is an approach worth mentioning. It is synergistic with traditional object-oriented techniques. While reuse based on components is becoming increasingly widespread, this does not mean that component-based techniques will replace object-oriented techniques. There are only subtle differences between the semantics of components and classes.