CST 502 Emerging Language and Programming Paradigms
Drafted by Dr. Tim Lindquist
Draft dated December 2010

Catalogue Description. Emerging programming languages, their design, description, implementation and advanced features; facilities for creating secure distributed applications including API design and common patterns, facilities supporting software services, naming and composition; language paradigms and the resulting implications on execution; run-time systems and storage management; approaches to compilation and execution.

Textbooks.
1. Programming Language Principles and Practice by KC Louden
2. Language manuals and on-line resources for programming languages, tools and projects.

Prerequisites. Students are assumed to have the ability to program in an object-oriented programming language, and to have taken a course in data structures, algorithms and analysis.

Coordinator. Dr. Tim Lindquist

Course Description. The course covers the principles of programming and programming languages using a project-based approach that is programming intensive. Various programming language paradigms are used and compared, such as imperative compiled to object-code, and compiled to byte-code, object-oriented, static, functional, scripting, dynamic and declarative languages. The course provides an introduction to language design (syntax, typing and semantics), implementation approaches (lexical and syntactic analysis), and run-time execution environments. The course covers advanced features of programming languages such as API patterns, naming, generics, assertions, typing, inheritance, just-in-time compiling, storage management and operator overloading. The course involves extensive programming in multiple programming languages, and learning to use unfamiliar languages and program development tools. Students develop oral and written communication skills through in-class presentations and writing assignments associated with programming projects. Students may be asked to develop programs in various languages such as Smalltalk, Python, Objective-C, C++, Java, C#, XSLT, Scheme, and LISP.

Topics.
1. Language paradigms and their impact on recognition and execution.
2. Emerging languages and their facilities; Implementation approaches and supporting infrastructure.
3. Language specification; syntactic and semantic descriptions; facilities for language extension through API's or programmer-defined replacement of pre-defined language structures.
4. Language recognition, compilation, incremental compilation, and execution; intermediate forms, byte-code, common intermediate forms, object code, linking, libraries, just-in-time compiling.
5. Models of execution and runtime environments for programming languages; storage management.
6. Advanced features and API usage, such as assertions, sockets, threads and thread-safe programming, mapping program structures to hardware, generics,
attributes, properties, type systems, inheritance, protocols, categories, query language extensions, and features supporting multi-language systems.

6. Program development tools and their features; build and install tools, integrated development environments, debuggers, testing tools, program testing and analysis tools.

Course outcomes. Students are able to:

1. Communicate, apply and evaluate the tools, activities, and artifacts involved in programming language design, implementation, and program execution.
2. Analyze a problem, identify and effectively communicate candidate languages and associated tools making up the design, development and execution environment for solving the problem.
3. Knowledge of multiple language paradigms, corresponding languages utilizing those paradigms, implementation approaches including runtimes for each, and analysis of their use for problem solving; Especially imperative byte-code compiled, imperative object-code compiled, static, object-oriented, functional, and declarative.

Sample Projects:
1. Research and re-design storage management in an Object-based (Java or C#, for example) to provide additional language features that allow finer control over runtime management of objects.
2. Create a provider implementation for JNDI that supports a new naming system, such as distributed object registration and naming, or a database driven directory system.
3. Create an intermediate language (triples/bytecode) for a simple object-oriented language. Construct an interpreter for the triples.
4. Create a threaded chat server and a corresponding client. Communication between the client and server should occur through stream-based sockets. Create a second chat client in a language different from the server.
5. Create a syntax analyzer and intermediate code generator for a simple imperative language.
6. XML manipulation project. Create an xml schema definition for source code documentation. Generate software to generate and parse corresponding xml documents, verify they adhere to the schema, and create the XSLT to convert the xml into html.