Can Domain-Specific Languages Be Implemented by Service-Oriented Architecture?

Domain-Specific Languages

A Domain-Specific Language (DSL) is a programming/modeling language that shields accidental complexity by uplifting the abstraction layer to a higher level.

A DSL introduces domain-specific constructs and notations to facilitate productivity, reliability, maintainability and portability.

Decision, analysis, design and implementation patterns have been identified to assist DSL developers in when and how to develop a DSL.

Example DSLs include:
- Robot language: An imperative DSL that controls a Lego® Mindstorms® (NXT) robot to move in different directions and distances.
- PPCea: An imperative DSL that controls parameter settings to balance an evolution process toward optimization and convergence.
- Feature description language (FDL): A declarative DSL to configure combinations of features.

Current Challenges

When implementing a DSL, several obstacles have appeared due to frequent need to represent changes in domain concepts. These obstacles are especially critical for DSLs following the interpreter and compiler implementation patterns.

- Extension/Evolution: When domain concepts change, then the lexical, syntactical and/or semantic domain constructs need to evolve. Yet, such evolution is tedious and error-prone. For example, a new domain statement or one new grammar production introduced will affect an existing DSL implementation at the lexical, syntactical, and semantic levels in different magnitudes.
- Interoperability: A DSL is usually implemented by one base language (e.g., Java). What if it is desired to implement a DSL in different base languages? How would these base languages communicate with each other?
- Tool Support: When a new DSL is introduced, corresponding DSL tools should be supported. Otherwise, the DSL will have fewer opportunities to be adopted. Yet, such a need requires a great amount of elaboration and promotion.

DSL Implementation Methodologies

There has been no DSL developed using SOA yet.

- AMMA is a platform to implement text-based DSLs using a Model-Driven Engineering approach that is focused on model transformations.
- The Generic Modeling Environment (GME) is a metamodelling toolkit for developing graphical DSLs. MetaEdit also provides similar functionalities.
- Six DSL implementation patterns are identified:
  - Interpreter/compiler patterns utilize the traditional compiler/interpreter interface.
  - Embedding patterns introduce new DSL constructs from an existing General-Purpose Language (GPL). (c) Preprocessor patterns translate DSL constructs into a base language; (d) Extendible compiler/interpreter patterns add DSL optimization rules and code generation in the existing compiler/interpreter of a GPL; (e) Commercial-off-the-shelf patterns utilize existing tools and/or notations for a specific domain; and (f) A Hybrid pattern is the combination of all of the above.
- There are many other implementation methodologies for DSL (e.g., Visual Studio DSL tools and MetaEdits+).

SOA approach for PPCea

PPCea utilizes if-else, while, assignment and DSL statements to dynamically control parameters of Evolutionary Algorithms on-the-fly.

- WS-BPEL is an executable language to specify the interactions among web services.
- It has various programming constructs to describe the execution flow of a business process.
- For a SOA-based DSL, WS-BPEL describes logical and issues that may emerge in a DSL program.
- WSDL defines a web service as a “software system designed to support interoperable machine-to-machine interaction over a network.”
- For a SOA-based DSL, a web service describes the semantics of a DSL statement.
- WS-BPEL expressions define Initialize, Select, Mutate, Crossover, Evaluate, Update and Entropy web services to adaptively obtain optimal solutions of an evolution process.

SOA Approach for FDL

- A Feature Description Language (FDL) is a declarative language that textually describes feature diagrams for domain analysis.
- The language introduces all-of, one-of, more-of and optional feature operations to explore all possible configurations along with requires, excludes, include and exclude constraints to reduce the possibilities.
- FeatureWS comprises the aforementioned eight operations, each of which consumes previously generated and current XML messages and returns a new combinatory XML message based on normalization, variability, expansion and satisfaction rules.
- CompleteWS prints out the final combinatory result of all possible alternatives.
- A preprocessing step is also needed to convert the above program to XML messages line-by-line.

Similar to the SOA-based PPCea:
- WSDL acts as a lexical analyzer and assist with syntax analysis.
- XML schema validates XML messages.
- WS-BPEL specifies DL semantics.
- WS-BPEL describes the execution flow of domain-specific web services to be consumed.

Conclusion

- SOA-based DSLs offer five implementation advantages:
  - SOA addresses the extension and evolution problems at syntactic and semantic levels. For new, existing extended, or evolved DSL core code can be dynamically modified by automatically generated WSDL files, and semantic evolution is achieved by introduction and/or composition of web services.
  - SOA offers interoperable communications among Web services implemented in different domains, which addresses interoperability concerns of DSL implementation.
  - WS-BPEL is a technology-neutral language that has been adopted by many vendors. It may reduce the effort to introduce tools for new or existing DSLs.
  - SOA offers improved modularity at the lexical, syntactical and semantic levels.
  - Lexical and syntax analyses adopt an interpreter or compiler-based DSL implementation are no longer needed in SOA-based DSLs.
- SOA-based DSLs may raise potential research interests to overcome the tradeoffs surrounding the flexibility of WS-BPEL grammars, WS-BPEL usability, bottlenecks on XML parsing time, and exposed domain-specific parameters.

Discussions

- Lexical Analysis and Symbol Table:
  - There is no need to perform lexical analysis: WSDL can be regarded as the lexical analyzer.
  - Symbol tables cannot be achieved easily. XML message passing between web services is an alternative.
- Syntactical Analysis
  - WS-BPEL specifications have defined grammars. Re-inventing SOA-based DSL grammars and parsers for PPCea and FDL (and even the Robot language) are not needed.
  - Yet, WS-BPEL’s great flexibility may be also misused and can result in potential pitfalls.
- Semantics and Type Checking
  - Domain-specific statements are wrapped as one or more web services.
  - Implementation of DSL web services is not much different except:
    - An internal commonly shared symbol table is no longer valid. Investigation on analyzing the scope of domain-specific parameters is needed - only those that will be needed by most web services will be encapsulated in XML messages.
    - There is a need to introduce efficient marshalling and unmarshalling algorithms to parse the aforementioned XML messages. JAXB is a more formal approach: an XML schema is used to validate and convert between object and XML instances. Conversely, SAX is a more casual but efficient way that processes XML as a stream and ignores tree construction.

SOA-based DSLs offer five implementation advantages:

- SOA addresses the extension and evolution problems at syntactic and semantic levels. For new, existing extended, or evolved DSL core code can be dynamically modified by automatically generated WSDL files, and semantic evolution is achieved by introduction and/or composition of web services.
- SOA offers interoperable communications among Web services implemented in different domains, which addresses interoperability concerns of DSL implementation.
- WS-BPEL is a technology-neutral language that has been adopted by many vendors. It may reduce the effort to introduce tools for new or existing DSLs.
- SOA offers improved modularity at the lexical, syntactical and semantic levels.
- Lexical and syntax analyses adopt an interpreter or compiler-based DSL implementation are no longer needed in SOA-based DSLs.
- SOA-based DSLs may raise potential research interests to overcome the tradeoffs surrounding the flexibility of WS-BPEL grammars, WS-BPEL usability, bottlenecks on XML parsing time, and exposed domain-specific parameters.