Requirements-Based Design Guidance: A Process-Centered Consistency Management Approach

Aaron G. Cass **Leon J. Osterweil**

Department of Computer Science
University of Massachusetts
Amherst, MA 01003-4610
+1 413 545 2013
{acass, ljo}@cs.umass.edu

Abstract

In previous work, we have argued that constraints on the relationships between requirements elements and design elements can be used to guide designers in making design decisions. We have also argued the importance of controlling which constraints to apply, when to apply them, and what responses to take on constraint failure in order to make this guidance effective. Our suggestion, to control the application of constraints with a formalized, executable process program, offers the benefit of not applying all constraints at all times and therefore not overwhelming designers with input, most of which is not applicable at all times. Rather, our approach offers the possibility of identifying and exploiting the most appropriate constraints at the most opportune times.

In this paper we present results of our experiments in applying this approach to a UML design based on requirements specified as use cases. The paper emphasizes two observed benefits of our approach. First, we note that our approach enhances traceability in that it facilitates control of the execution of constraints. Second, we note that our approach enables the specification of consistency rules for design milestones, thereby guiding designers without sacrificing important latitude and flexibility during the design process.

1 Introduction

Software systems consist of multiple development artifacts, with complex relationships between them – the design must satisfy the requirements and be implementable, the code must implement the design and be testable, and the tests must exercise the code and test according to the requirements. An effective model of a software system is one in which this rich web of relationships exists between artifacts and in which well-formedness constraints on those relationships are satisfied. Our ongoing research is concerned with helping software developers to manage these relationships and constraints and therefore to build better software.

In particular, we are interested in developing technologies to provide guidance to designers in the development of good designs. Because designs must satisfy the given requirements of a software system, we aim to provide guidance for the designer to make effective design decisions based on those requirements. In previous work [5], we have argued that constraints on the relationships between requirements elements and design elements can be used to guide designers in making design decisions. By controlling constraint application with a formalized, executable process program, we can control which constraints to apply at which times and what response should be taken if any of the constraints fail. We have argued that controlling the application of constraints has the benefit of not overwhelming the designer with reports of the failures of many inapplicable consistency constraints.

In this paper, we present the results of our further experimentation toward a design environment that supports designers by controlling the application of consistency constraints. In our latest experiments, we have used the approach for a UML [12] design process which guides designers to produce class diagrams (design elements), given use cases with activity diagrams (requirements elements). From this experimentation, we observe that controlling constraint application with a formalized process program has many benefits for designers. In this paper, we highlight two such benefits. First, we note that our approach can be used enhance requirements-design traceability because the presence or absence of traceability relationships can be specified by constraints to be applied. Second, we note that our approach enables the specification of design milestones to guide designers without overwhelming them with superflu-

ous input and while not sacrificing important latitude and flexibility during the design process.

2 Related Work

There is a long history of proposing processes, both formally and informally defined, aimed at arriving at high-quality designs. For example, the design patterns community [7] suggests first deciding what will vary in the system and then choosing a pattern that encapsulates in such a way as to allow for that variability. This and other approaches promise to help designers to arrive at high-quality designs. However, we are interested in two objectives that are generally not addressed:

1. Active guidance, during development, for designers to follow the process
2. Active guidance, during development, for ensuring that the design, whatever its other qualities, actually satisfies the intended requirements – that the system model has a high level of desired consistency

To pursue these objectives, we are interested in managing consistency of system models with respect to some consistency rules. However, as Emmerich, et al. [6] argue, it is not desirable to enforce consistency (in their case, standards compliance) at all times during development. They propose an approach that controls when properties are checked. Our approach is similar in that we control consistency checking with a process model. However, their approach is based on watching for and reacting to trigger events, while our approach uses a proactive process specification. Another difference is in the flexibility of reaction to constraint failures – they allow pre-programmed reactions to be performed by the system, while we allow any action which the process programmer can assign to any participant in the process, human or automated.

Others have proposed approaches that use consistency rules to provide guidance to designers while they work. A notable system is Argo/UML [10], a UML model-editing tool in which critics [11] check, after each addition to a system model, that certain rules are satisfied. If any of the rules are not satisfied, an item is added to the user's prioritized to-do list indicating the problem and offering suggestions about how to fix the problem. One problem with the Argo/UML approach is that there is no automatic control over which critics are active at which times. This generally results in many to-do list items identifying shortcomings of the current system model, many of which are errors of omission for which the user has not previously had an opportunity to add the model elements that would have satisfied the consistency rules. There is, however, manual control available to turn critics on and off.

Argo/UML's critics are written in Java for maximum flexibility. Others, however, have proposed formal languages to be used to specify consistency rules. The standard constraint language for UML models is the Object Constraint Language (OCL) [2], which allows the identification, using path selection operations, of sets of model elements and the specification of constraints on relationships between those elements using first-order logic. This has been used in the definition of the semantics of UML. However, because OCL is based on first-order logic, transitive closure is not available, thereby reducing the potential usefulness for specifying consistency rules.

xlinkit [8] is similar in that it allows the identification of sets of elements, though it works on XML-encoded documents and uses XPath [15] expressions for path selection. It is also based on first-order logic[1] but adds transitive closure. Since many UML tools, including Argo/UML will output UML models using XMI [9], an XML-based standard encoding of UML models, xlinkit can be used to specify and check consistency rules for UML system models.

3 Our Approach

Our approach is to define software design as a formally-defined process program. For this purpose, we use our process language, Little-JIL [14, 13], which defines processes in terms of a hierarchy of steps to be performed. In our approach, we attach consistency checks to specific steps in the process program, usually post-requisites of other steps – at those points the consistency rules can be checked on the appropriate development artifacts. If any of the artifacts fail the consistency checks, an exception will be thrown for which there can be a response programmed into the process program.

An Example

In recent experimentation, we have applied this approach to create a process for developing a UML design, with the intention of providing an environment to guide a designer in the development of a good design – one that is consistent with the requirements for the software system. To explain the approach, we now give an example from those experiments.

In the example scenario, the designer's goal is to develop a design for a validated graph editor. The graph editor will allow users to create nodes, annotate them, and connect them with edges which can also have annotations. Throughout, the graph editor should flag those portions of the graph that are invalid with respect to certain rules and disallow the creation of invalid graphs.

The project starts with the creation of a set of use cases whose functional requirements are described by activity di-

[1] xlinkit actually restricts consistency rules to start with the quantifier $\forall$, but this does not restrict the rules we can express.

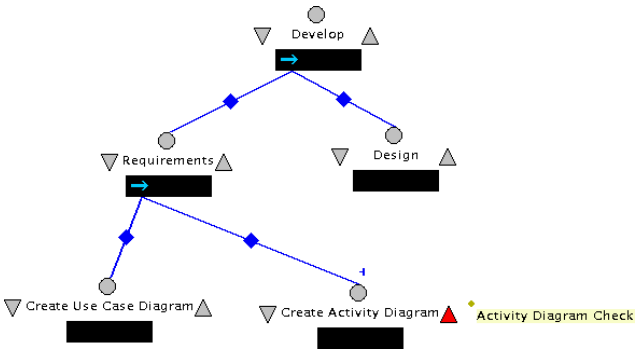

Figure 1: Example Little-JIL Process Program

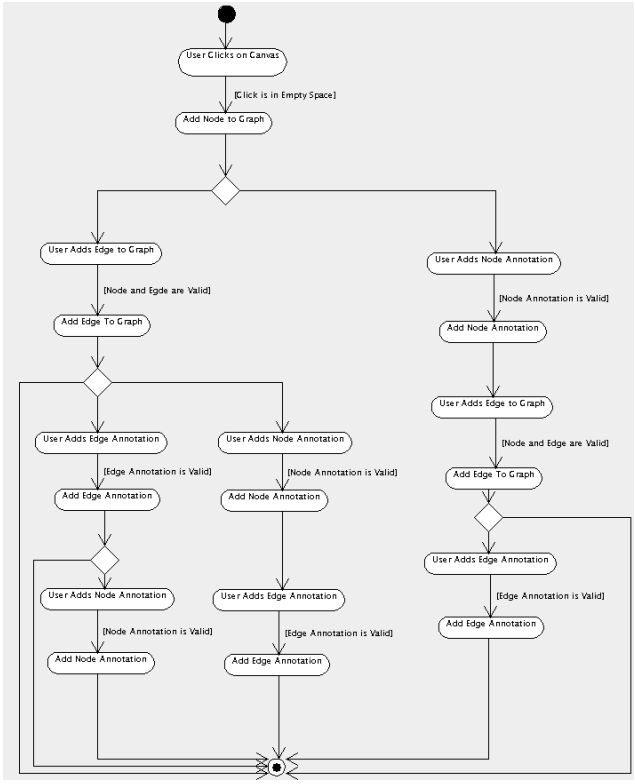

Figure 2: Example Use Case Activity Diagram

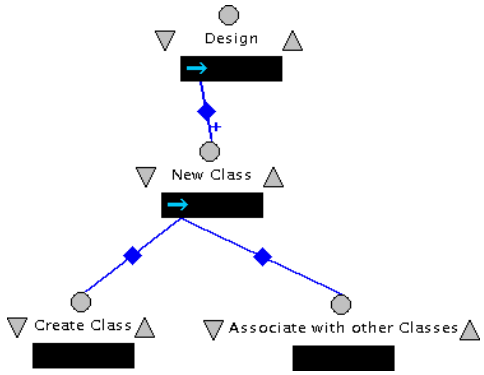

Figure 3: Example Little-JIL Process Program(continued)

agrams. Figure 1 shows the first part of the process to accomplish this. The process is a collection of steps (the black bars) that are hierarchically organized and, as indicated by the right-pointing arrow in the Develop and Requirements steps, sequenced from left to right. The plus sign above Create Activity Diagram indicates that this step will be performed 1 or more times, at the discretion of the participants in the process. A darkened triangle after a step indicates a post-requisite, which is a reference to a separate step that must be executed upon completion of the step. Notice in the example process that Activity Diagram Check is a post-requisite of Create Activity Diagram. In this example, the intention is to have Activity Diagram Check perform consistency checks for a single diagram. These will be intra-diagram checks which will of course be needed to produce a good, complete model of the system. In the rest of the paper, however, we will focus on inter-diagram consistency checking.

Figure 2 shows an activity diagram for a use case that might be created by the process described so far. This activity diagram is for the use case describing how the user might add a node to an existing graph. This use case describes the case where a user adds a node and then connects it to the rest of the graph.

Figure 3 shows an elaboration of the Design step from the process in Figure 1. In this part of the process, the designer creates a class diagram for the software system. Again, at specified points in the process, consistency checks can be performed, with the aim of producing a high-quality design by the end of the process. Figure 4 shows a possible incomplete class diagram that might be created with this process.

We will use this example process program to highlight, in the following sections, two benefits of using this approach.

Improving Traceability

Clearly, some of the consistency rules that govern whether the design is a good one or not will rely on the existence in

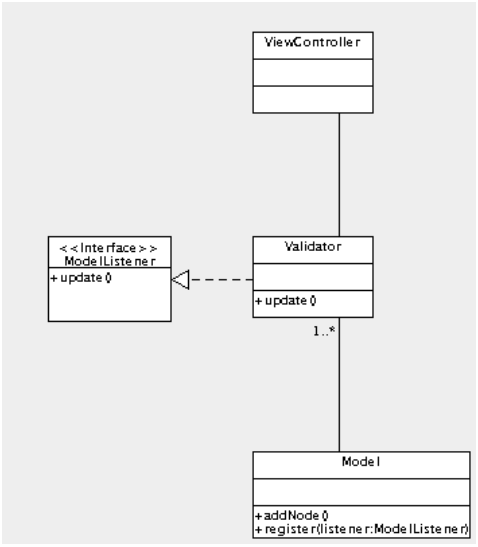

Figure 4: Example Class Diagram

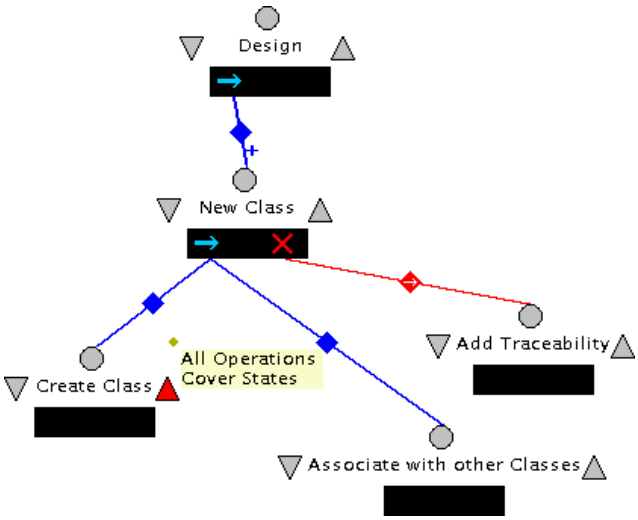

Figure 5: Example Little-JIL Process Program(revised)

the model of certain traceability information. It will therefore be profitable to require the traceability information to be entered in the model as part of the development activity. Our approach of checking consistency rules at specified points in the process can be used to accomplish this.

For example, after each class is created (as a post-requisite on the Create Class step), we can check that each method of the class is related to an action state in one of the activity diagrams, via an $\rightarrow_{implements}$ relationship. The rule can be represented by the following consistency rule[2]:

$$\begin{aligned} &\forall o \in operations : \\ &\quad \exists s \in actionStates : \\ &\qquad o \rightarrow_{implements} s \end{aligned} \tag{1}$$

If this consistency rule is satisfied, this will ensure that every class we create is related to some element of the requirements specification. In the example class diagram in Figure 4, if the addNode method on Model has an $\rightarrow_{implements}$ dependency with the Add Node to Graph state from Figure 2, then addNode will satisfy the consistency rule from Rule 1, but the register method does not. In the process program, we can respond to this exception by requiring the designer to add this traceability information. Figure 5 shows a revision of the Design step that indicates where these checks can be performed and how to respond to failure. The Add Traceability step is a handler for the exception which might be thrown by the All Operations Cover States step (a post-requisite of Create Class). The right-pointing arrow on the edge to Add Traceability indicates that once the traceability information is added, the designer can continue on to the next step, in this case Associate with other Classes.

We would also, in the course of design, like to ensure not only that all operations respond to some portion of the requirements, but also that every requirement is covered. This can be formalized with the following consistency rule:

$$\begin{aligned} &\forall s \in actionStates : \\ &\quad \exists o \in operations : \\ &\qquad o \rightarrow_{implements} s \end{aligned} \tag{2}$$

Of course, this rule will not be satisfied during most of the development of the design – not until all classes are declared is it reasonable to expect that all requirements have been covered. Figure 6 shows a further revision of the Design step to check conformance only after the class diagram is complete. Notice that to accomplish this, we have added an extra scope, called Create Class Diagram, so that the possible exception can be contained within the Design step's hierarchy. In this revision, we have added a post-requisite to the Create Class Diagram step to check conformance with the consistency rule. By waiting until after the Create Class Diagram step is completed to apply this rule, we avoid checking that the requirements are all covered before the designer has had a chance to give the full design.

As was the case in the previous rule, a response to a failure of this rule will throw an exception which can be handled by the rest of the process program. In this case, we give

[2]In this and the following formalizations of consistency rules, we assume two sets have been defined: $operations$ is the set of all operations (methods) on classes in the model and $actionStates$ is the set of all action states in activity diagrams.

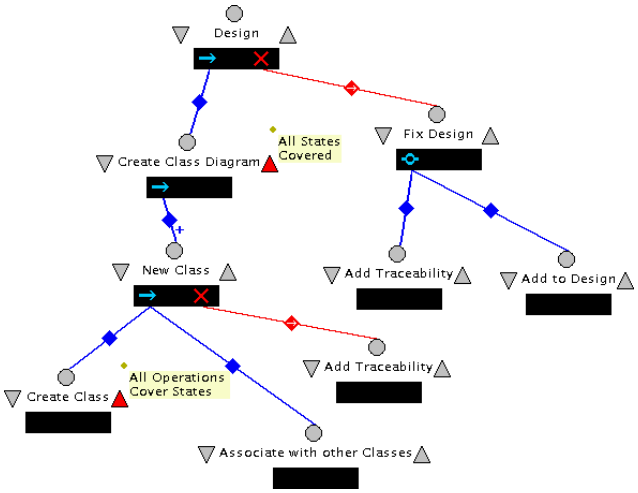

Figure 6: Example Little-JIL Process Program(revised)

the designer a choice (as indicated by the circle and line icon on the Fix Design step) between adding traceability information and adding to the design to cover the rest of the requirements.

Milestones

While Rules 1 and 2 can be used in a process program to ensure that all operations cover requirements and that all requirements are covered in this way, they do not ensure that the relationship between operations and states is one-to-one. The following consistency rule can be used to check that states are not covered by multiple operations:

$$\begin{aligned} &\forall s \in actionStates; o1, o2 \in operations : \\ &\quad o1 \rightarrow_{implements} s \wedge o2 \rightarrow_{implements} s \\ &\quad\quad \Rightarrow o1 = o2 \end{aligned} \tag{3}$$

Of course, this is not a consistency rule that would be required of the finished system model – a designer might create several operations that together implement a single action state from the requirements. However, this rule and ones like it can be used as milestones.

In a consistency management approach in which consistency rules are always applied (like the default behavior of Argo/UML) or only applied at one time (such as when the user saves a model to disk), there is only one set of rules to apply and rules that are not applicable to finished models will not be included in the rule set. However, because we allow the specification of which rules to apply when, rules can be used as milestones during a process even though enforcement is not desired on the final product.

In particular, in designing a system for guiding novice designers to make effective decisions, it might be desirable to guide the designer to such intermediate states. With these states marking milestones in the process we can guide the designer to create a high quality final solution. By setting these milestones, we narrow the design requirements and therefore give the novice more direct guidance about what to do instead of leaving the designer to make intelligent decisions in the face of many, perhaps overwhelming, options.

In the case of the $\rightarrow_{implements}$ relationship between requirements and design, an intermediate system design in which the design has a single method for each state in the requirements might be a good starting point for elaborating the design. After that point in the process, the consistency rules can be relaxed (in this case, by removing Rule 3 from the rule set).

4 Experience and Future Work

Our experiments with this approach has, to this point, been mostly concerned with issues of feasibility. Thus we have been attempting to answer questions such as the following:

- Is it feasible to control application of constraints using a formalized process program?
- What effect, if any, does adding consistency checking have on a design process?

Feasibility

To investigate the feasibility of the approach, we are building an infrastructure based on Juliette [3, 4], the run-time system for Little-JIL. In an Little-JIL process program, every step has a specification of an execution agent which is responsible for carrying out the step. In the example in Figure 1, the Requirements step would specify that it needs a requirements engineer, while the Design step would specify a need for a designer. At run-time, Juliette reads in a Little-JIL program and determines which agents to bind to step instances and then assigns those step instances to the agents in the sequence specified by program. Agents interact with Juliette by way of an agent API or, in the case of human agents, a user interface written to this API.

In order to integrate consistency checking, we have developed an automated agent that uses the xlinkit [8] consistency checker to check the appropriate rules on the appropriate UML model, both of which are passed as parameters to the agent. The agent specifications on the the post-requisites in Figures 1, 3, 5, and 6 indicate a need for this new type of agent. At run-time, Juliette assigns these post-requisites to the new agent, which executes the xlinkit check. If xlinkit finds an inconsistency with respect to any of the rules, the agent throws an exception which can be handled in the rest of the process program, as shown in the process examples in this paper.

In order to use xlinkit, we had first to write the rules using xlinkit's rule language and get our UML models in an XML

form for xlinkit to use them. We are using Poseidon [1], a commercial version of Argo/UML, as our UML-modeling tool because it is freely available and because it will output directly to XMI. We should note, however, that because Poseidon does not fully support UML's dependency associations, the consistency rules are slightly more complex than presented in this paper. We have had to represent the relationships with tagged values, which Poseidon does support, and then use an extra quantifier in the rules to select the appropriate tag. In the design example, $a \rightarrow_{implements} b$ is represented as a tag on a named "implements" and with value b. Using tagged values gives us the flexibility to define any dependency we wish between any two elements in the model, not just those elements that a particular UML model-editing tool allows.

Our experiments to date indicate that the approach is feasible using Little-JIL and an infrastructure can be built to turn the Little-JIL process into a design environment that does automated consistency checking at the appropriate times.

Effect on the process
As can be seen by comparing the process models in Figures 3 and 6, the addition of consistency checks sometimes has the effect of requiring an extra level of scope for the exception handlers so that a step can continue to contain failures. By adding a scope in this way, the change is localized, because the change is not revealed above the new scope. In the example, the Design step does not expose that anything is different, because its interface does not change.

While this has the disadvantage of requiring that new scopes be added during the development of the process, the disadvantages can be minimized by localizing the required changes. The advantage of being able to automate effective use of consistency rule checking seems to out-weigh this disadvantage.

Future Experiments
While our experiments so far indicate that the approach is feasible, we need further experiments to determine the usefulness of the approach for providing design guidance. Our plan is to continue to develop the infrastructure to better integrate system modeling with the consistency checking and the process. We can then carry out experiments in which designers use the approach to design a piece of software.

We are particularly interested in determining what level of granularity of action is appropriate for the process. Should steps be broken into many sub-steps to give designers fine-grained guidance, or should steps be higher-level to give designers freedom? We expect this will depend on the experience of the designers. Clearly, the answers to these questions have implications for the tightness of integration needed between the modeling tools and the process run-time environment.

5 Conclusions

In this paper, we have presented an approach to design guidance using consistency management under the control of a formalized process program and we have presented details for our infrastructure which will support this approach. We have given examples from our experiments that highlight key benefits of the approach.

We have demonstrated how the approach might be used to ensure that traceability information, which might be used in later consistency checks, is present in the model. We have also shown that the approach can be used to establish milestones to guide a designer even if enforcement of the rules for the milestones are not desirable in the finished product.

The approach represents one way to help software developers effectively manage the rich web of relationships between artifacts and well-formedness rules on those relationships.

Acknowledgements

The authors would like to thank Christian Nentwich and Anthony Finkelstein for their help obtaining and using xlinkit, and for their quick response to problems and suggestions.

We would also like to thank David Jensen for the inspiration for the graph editor example system – it is similar to the project for the undergraduate software engineering course that one of the authors recently taught and for which Professor Jensen acted as the customer.

This research was partially supported by the Air Force Research Laboratory/IFTD and the Defense Advanced Research Projects Agency under Contract F30602-97-2-0032 and by the U.S. Department of Defense/Army and the Defense Advance Research Projects Agency under Contract DAAH01-00-C-R231. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied of the Defense Advanced Research Projects Agency, the Air Force Research Laboratory/IFTD, the U.S. Dept. of Defense, the U. S. Army, or the U.S. Government.

References

[1] Poseidon UML Community Edition. www.gentleware.com.

[2] *Object Constraint Language Specification*, Sept. 1997. version 1.1.

[3] A. G. Cass, B. S. Lerner, E. K. McCall, L. J. Osterweil, S. M. Sutton, Jr., and A. Wise. Little-JIL/Juliette: A process definition language and interpreter. In *Proc. of the* 22^{nd} *Int. Conf. on Soft. Eng.*, June 2000. Limerick, Ireland.

[4] A. G. Cass, B. S. Lerner, E. K. McCall, L. J. Osterweil, and A. Wise. Logically central, physically distributed control in a process runtime environment. Technical Report 99-65, U. of Massachusetts, Dept. of Comp. Sci., Nov. 1999.

[5] A. G. Cass and L. J. Osterweil. Design guidance through the controlled application of constraints. In *Proc. of the Tenth Int. Workshop on Soft. Specification and Design*, Nov. 5–7, 2000. San Diego, CA.

[6] W. Emmerich, A. Finkelstein, C. Montangero, S. Antonelli, S. Armitage, and R. Stevens. Managing standards compliance. *IEEE Trans. on Soft. Eng.*, 25(6):836–851, 1999.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[8] C. Nentwich, L. Capra, W. Emmerich, and A. Finkelstein. xlinkit: A consistency checking and smart link generation service. *ACM Trans. on Internet Tech.*, 2002. To appear. Available at www.cs.ucl.ac.uk/staff/A.Finkelstein/papers.

[9] Object Management Group. *OMG XML Metadata Interchange (XMI) Specification version 1.2*. Object Management Group, Jan. 2002. www.omg.org/technology/documents/formal/xmi.htm.

[10] J. E. Robbins, D. M. Hilbert, and D. F. Redmiles. Argo: A design environment for evolving software architectures. In *Proc. of the Nineteenth Int. Conf. on Soft. Eng.*, pages 600–601, May 1997.

[11] J. E. Robbins and D. F. Redmiles. Software architecture critics in the Argo design environment. *Knowledge-based Systems*, 11(1):47–60, 1998.

[12] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.

[13] A. Wise. Little-JIL 1.0 Language Report. Technical Report 98-24, U. of Massachusetts, Dept. of Comp. Sci., Apr. 1998.

[14] A. Wise, A. G. Cass, B. S. Lerner, E. K. McCall, L. J. Osterweil, and S. M. Sutton, Jr. Using Little-JIL to coordinate agents in software engineering. In *Proc. of the Automated Software Engineering Conf.*, Sept. 2000. Grenoble, France.

[15] World Wide Web Consortium. *XML Path Language (XPath) version 1.0*. World Wide Web Consortium, Nov. 1999. http://www.w3.org/TR/xpath.