In Proceedings of the 34th Hawaii International Conference on System Sciences, Maui, Jan 3-7 2001, IEEE CS Press.

Software Architecture Modelling, Analysis and Implementation with SoftArch

John Grundy

Department of Computer Science, University of Auckland
Private Bag 92019, Auckland, New Zealand
john-g@cs.auckland.ac.nz

Abstract

Good software architecture design is crucial in successfully realising an OOA specification with an appropriate OOD model that meets the specification's functional and non-functional requirements. Unfortunately most CASE tools and software architecture design notations do not adequately support software architecture modelling and analysis, nor integration with OOA & D models. We describe SoftArch, an environment which provides flexible software architecture modelling using a concept of successive refinement. SoftArch also provides extensible analysis tools enabling developers to analyse their architecture model properties. This paper overviews the motivation for SoftArch, its modelling and analysis capabilities, and its integration with various analysis, design and implementation tools.

1. Introduction

Many software modelling notations and tools have been developed [6, 7, 12, 15], and there has been an increasing emphasis on software architecture modelling in addition to OOA & D modelling in CASE tools. Various approaches have been tried, including those of UML [3], PARSE [15], JViews and aspects [8, 10], tool abstraction [7], and Clock [6, 22]. Support tools include Rational Rose [18], JComposer [10], PARSE-DAT [15], ViTABaL [7], SAAMTool [12] and Argo/UML [19].

Most of these systems provide partial software architecture modelling solutions, with only some aspects of architecture modelling supported e.g. basic structure, limited dynamic behaviour and event models, dynamic process creation etc [11, 14]. Few provide adequate analysis tools to help developers reason about their models and ensure OOA requirements are met and all software architecture components are refined to suitable OOD abstractions [19, 11]. Few support OOD and/or implementation code generation from architecture-level abstractions, and few support reuse of previously developed models and patterns [22, 19]. Almost none allow new architecture abstractions and analysis tools to be added, and most have poor or no integration with OOA, design and implementation tools.

We describe SoftArch, a new, extensible environment using new approaches to software architecture modelling, analysis, design generation and tool integration. SoftArch uses an extensible meta-model of architecture abstractions. Architects use a flexible, extensible visual notation based on allowable abstractions to describe and refine software architecture models, including links to OOA and design objects and classes as appropriate. A collection of extensible "analysis agents" constrain, guide and advise architects as they build and refine their architecture models. SoftArch has been integrated with several OOA, design and implementation tools, as well as a process management environment, using a variety of tool integration techniques.

The following section presents a motivation for our work developing SoftArch. We then overview the environment's capabilities, and in the following sections describe its software architecture modelling, refinement and analysis support. A brief discussion of SoftArch's implementation and architecture is presented, and we conclude with a summary of the contributions of this work and directions for future research.

2. Motivation

Software architecture development has become an increasingly important part of the software lifecycle, due to the increasing complexity of software being constructed [1, 3, 21]. Software developers need to carefully describe and reason about the architectures of complex, distributed information systems, which are often comprised of a mix of new and reused components. A good, extensible and maintainable architecture often makes the difference between successful and failed projects. Much more time tends to be spent on architecture development than previously, and many more options exist for developers [1].

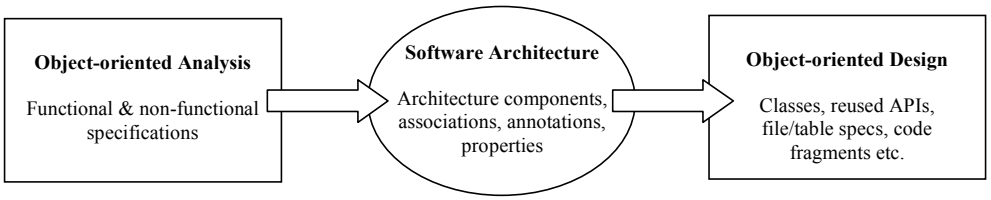

Figure 1. Transformation of OOA model to OOD model via Software Architecture.

This includes a wide range of software architecture styles/patterns [5], technologies to realise system architectures [20, 17], and existing system architectures to integrate new systems and components with.

We view the role of software architecture as a key mechanism for supporting developers in successfully developing an OOD model to describe a system implementation that satisfies a given OOA specification (functional and non-functional requirements). Figure 1 illustrates this development process and relationships between OOA, software architecture, and OOD and implementation-level software artefacts. In addition, often existing designs and code must be reverse engineered into higher-level architectural models, which themselves may need to be reverse engineered into OOA specifications. Software architecture models typically need to capture high-level characteristics of a system, down to OOD-level system organisation [21, 14, 6].

Most existing software architecture notations and support tools don't adequately support architecture modelling, refinement, analysis and OOA/D linkage [11, 14]. This motivated us to develop the SoftArch environment. Originally this was to be an extension of our existing OOA/D/P tool JComposer [10]. However, we developed SoftArch as an independent tool and integrated it with not only JComposer, but also a range of other development tools.

3. Overview of SoftArch

SoftArch provides visual software architecture modelling support along with an extensible meta-model and development processes. A collection of extensible analysis agents guide, advise and/or constrain architects, and a set of reusable templates allow reuse of a variety of software architecture refinements. Figure 2 illustrates these basic SoftArch capabilities.

A key concept is the notion of refinement in SoftArch of high-level architectural model components into successively more detailed and lower-level models. Properties of high-level architectural components constrain the kinds of refinements and properties at lower levels of detail. An extensible meta-models of possible types of architectural components, relationships and properties constrain possible modelling architectural entities.

Analysis agents, controlled by software architects and enacted process models, monitor architecture model changes and advise architects on model correctness and quality. Analysis agents can act as constraints, disallowing invalid actions; can act as "context-sensitive advisors", giving immediate feedback as an architect works; or can be run batch-style to analyse properties or part or all of a model.

Import/export tools support linkage between SoftArch and OOA, design and implementation tools. OOA models allow software architects to capture functional and non-functional requirements in SoftArch and ensure software architecture models meet these, or at least are annotated with this information. OOD models and some code fragments (implementing socket protocols, database access, ORB API calls etc.) can be exported from bottom-level architecture components. Reverse engineering of OOD models into SoftArch allows developers to abstract higher-level architectural models from their code, and to ultimately export OOA specifications into CASE tools.

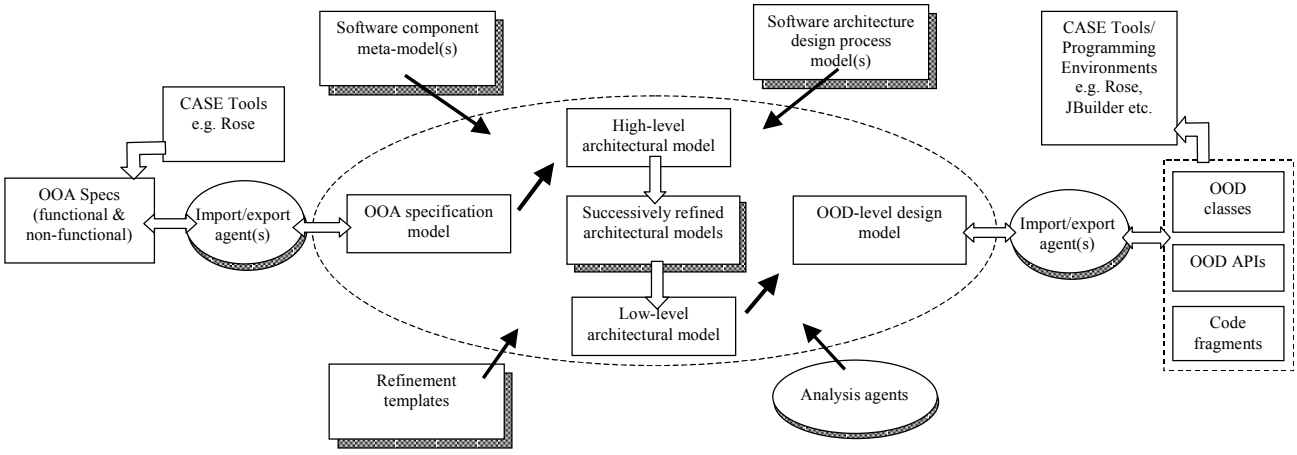

Figure 2. Overview of SoftArch architecture design modelling and analysis approach.

4. Software Architecture Modelling

4.1. Example Application

Figure 3 shows an example application and a possible (high-level) software architecture for this system. This is a video store library, with on-line customer search interface, in addition to corporate database with in-house forms, reports and batch processes for staff to use. We use this as an example in the following illustrations of SoftArch in use.

4.2. Software Architecture Modelling

Initially when developing such a system a software architect will import an OOA functional and non-functional specification from a CASE tool, or enter this information themselves. They then sketch out a high-level model (or copy and modify a suitable template model, if one exists from a previous project), ensuring the general characteristics of this model meets the OOA specification. They then refine this high-level model, successively adding more detail, and then generate an OOD model which will be further refined and implemented using a CASE tool and programming environment. This basic process is illustrated by the process model in Figure 4 (1). If part of a system exists, the software architect would import its OOD model and successively abstract software architecture components from it.

To represent software architecture models SoftArch uses a concept of architecture *components*, *associations* between components, and *annotations* on components and associations. Software architecture component abstractions include generic architecture entities, processes, data stores, data management processes (e.g. database servers), machines and devices, and OOA and OOD-level classes. Associations include generic architecture component associations, data usage associations, event notification/subscription, message passing, and process synchronisation links. Annotations include data used, events passed, messages exchanged, protocol used, caching, replication and concurrency information, process control information, ports and so on. Each of these architectural entities can have various properties specified. Properties include information such as services, security approaches, data size, transaction processing speed, data, message and event exchange details, and so on. Property values may be simple numbers or enumerated values, strings or value range constraints.

Visual views, along with property value dialogues, are used by architects to view and modify their architecture models. A set of meta-model elements describe available types of components, associations, annotations and properties.

Figure 4 (2) shows a high-level view of the video library architecture in SoftArch. The architect has represented the parts of the system as three "processes" – "staff client applications", "customer applets" and "servers". The staff applications are connected to the servers via a LAN association, the customer applets via an internet association. Two annotations indicate the staff applications use SQL commands and the applets a custom protocol to communicate with the servers.

The designer can have components shown in various ways (e.g. ovals for processes, squares for data management, cylinders for data storage etc.). Associations can be shown as lines, "bus"-style icons or network representations. Annotations include a name and symbol representing data, messages, events and caching.

Each component has a dialogue used to view and specify a unique name, component type, appearance configuration values, property names and values, associations, annotations and refinement information. Figure 4 (3) shows an example of such a dialogue. Meta-model elements available for use by an architecture model can be viewed using a visual notation and modified to change the available architectural abstractions.

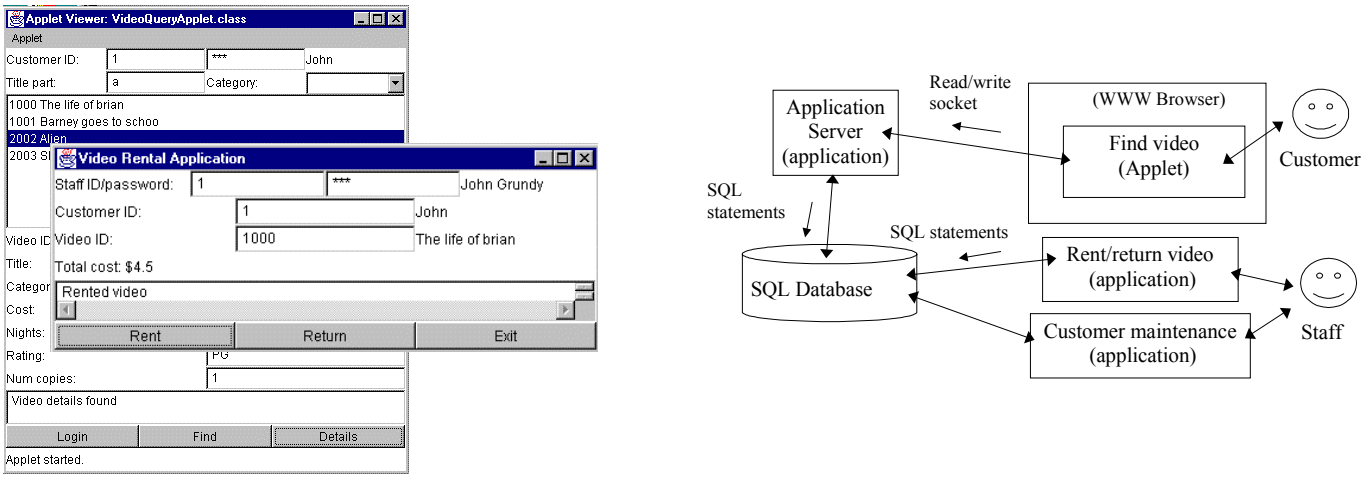

Figure 3. Example system for which to design a Software Architecture.

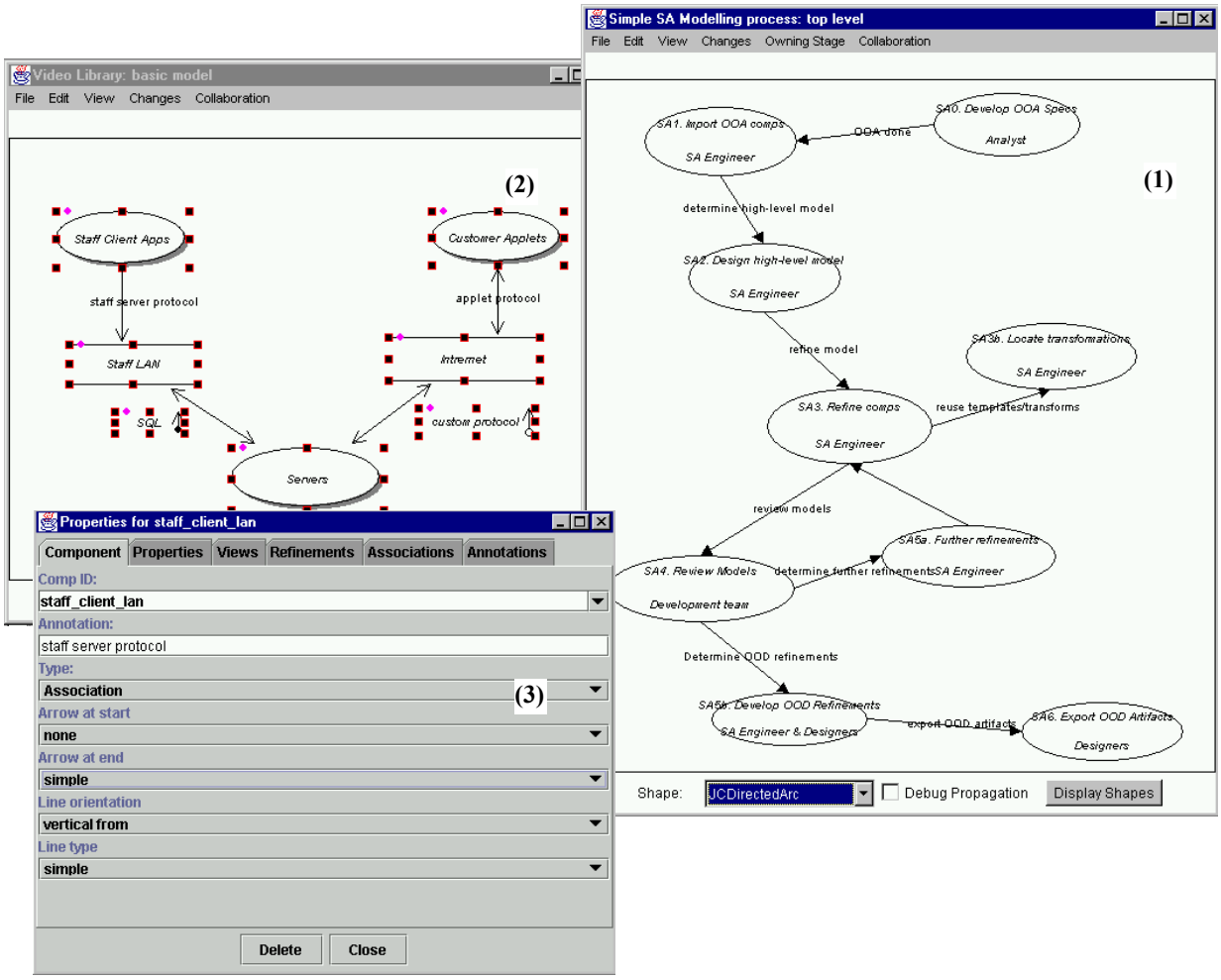

Figure 4. A high-level architectural model, component specification sheet and basic modelling process.

4.3. Refinement

There are three ways to refine a software architecture model in SoftArch: enclosing components, adding sub-views, and specifying explicit refinement links. Figure 5 illustrates each of these, along with an example of refinement information in a dialogue.

In diagram (1), the "Staff Client Apps" from Figure 4 has been refined by creating a sub-view for it, All components in this sub-view are refinements of the higher-level architecture component which owns the sub-view. A component may have several sub-views, with refined components shown in more than one sub-view. In this example, "Staff Client Apps" is refined to "customer maintenance", "video rent/return" and "video maintenance" processes. An annotation indicates that SQL commands are exchanged via the LAN with the servers.

In diagram (2), the "servers" component has been refined by using it to enclose other components. These include "http server", "application server", "rdbms_server" and "tables". Various associations have been specified both between enclosed components and between other components of the architecture and refinements of the "servers" component. All enclosed components, associations and annotations are refinements of the "servers" component.

In diagram (3), several architecture components, on the left hand side, have been refined to OOD-level class components on the right hand side. This was done by the use of explicit refinement links being added by the architect. In this example, "video query applet" is implemented by a "VideoQueryApplet.java" class, "application server" by "VideoQueryServer.java" and "VideoQueryServerThread.java" classes, and the connection between client and server implemented using sockets ("java.net.*" API). OOA-level classes and services can be refined to software architecture components in a similar way to indicate the analysis-level components architecture abstractions are being used to realise.

The dialogue in Figure 5 shows information stored for each refinement relationship, including unique name, abstract and refined components, and rationale for the refinement.

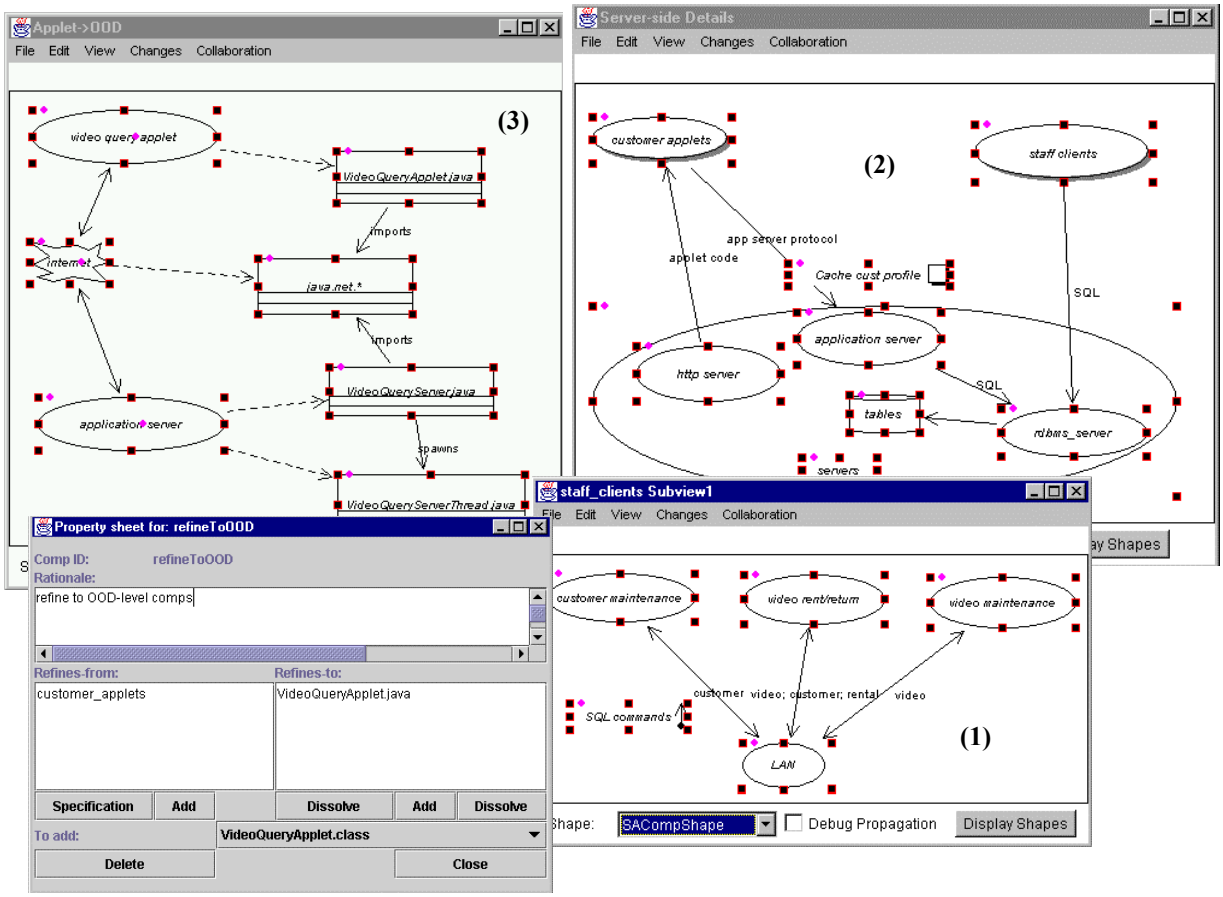

Figure 5. Examples of architecture refinement.

4.4. Templates

Many refinements are often reused when developing software architectures. For example, the "servers" component refined to http server, application server and RDBMS server as shown in Figure 5 (2) is a common refinement for simple e-commerce applications with Java applets. Thus we want to allow software architects to reuse such refinements on multiple projects, and package useful refinements for such reuse.

SoftArch allows architects to copy refinement views to create "templates", where one or more components are refined into the components and associations described by the template. Architects can then select an appropriate template and have SoftArch copy this into their project, automating linking of abstract components to new refined components copied from the template. Copied components and refinement links can then be modified if necessary by the architect. Changes to templates or copied refinements can be propagated back to one another using version merging support (a similar mechanism we developed for process model templates is described in [9]).

5. Software Architecture Analysis

Supporting modelling of software architectures and refinements is not sufficient to enable software architects to produce quality, consistent architecture models for complex systems. Software architecture analysis tools are also needed, including support for checking such things as: all components are linked to others, all components are suitably refined, all components are realised by OOD-level classes and are ultimately refined from OOA-level specifications, sensible and consistent associations and annotations have been used, valid property values have been set, provided and required services between linked components are met, and the model adheres to various "best practice" guidelines.

SoftArch provides an extensible set of analysis agents. These can be run as constraints, which fire whenever an architecture model is modified and inform the user immediately if an invalid action is attempted. They can be run as design critics, which monitor changes to the architecture model and report prioritised exceptions, poor choices, incompleteness or suggest possible improvements, in a non-intrusive way. The architect can review these from time to time and correct their model as they desire. A final approach is to have one or more agents run in batch mode over part or all of the architecture model. All exceptions they detect are presented in a report listing. Some agents may provide options to automatically correct the architecture model to correct problems, which the architect can choose to invoke.

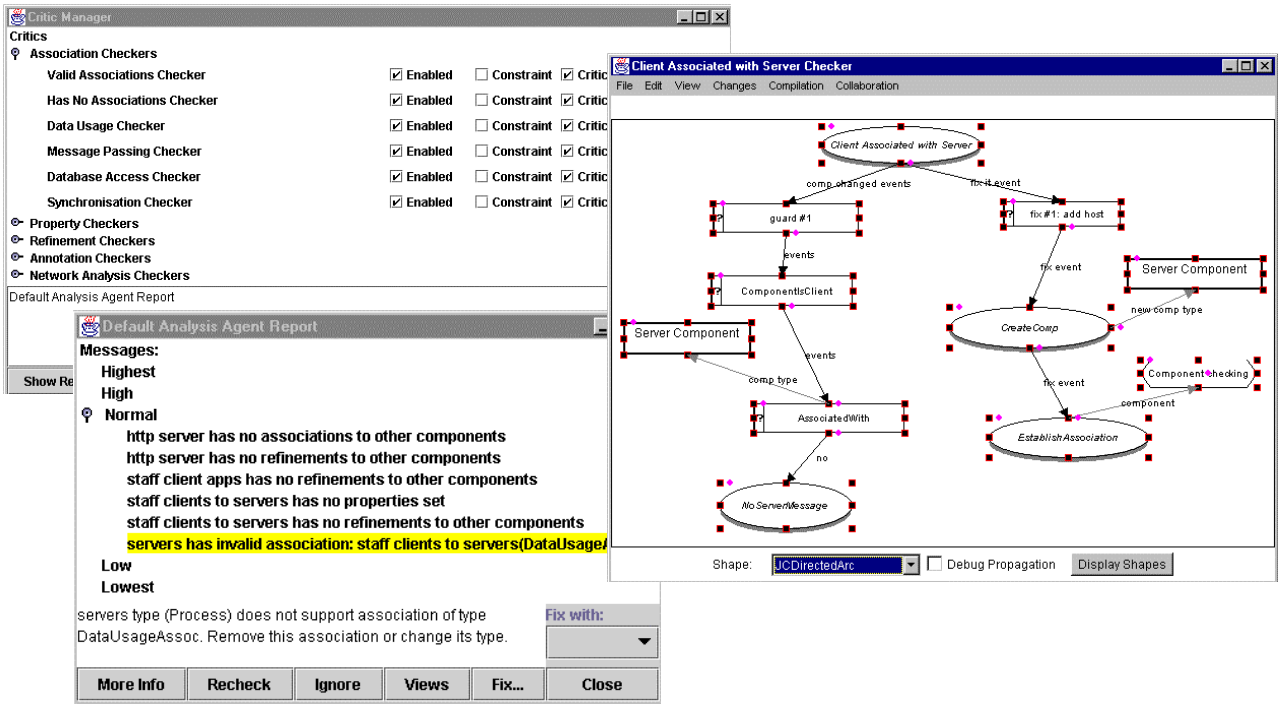

Figure 6. Analysis agent control, reporting and visual specification.

Agents are controlled by an analysis agent manager, as shown in Figure 6 (1). The architect can turn any agent on/off, change its priority, and change its detection mechanism (constraint, critic or analyser). The analysis agent manager organises agents by categories, and all agents in a category can be reconfigured at once by changing the category properties.

Agents running as constraints report detected exceptions using a dialogue box opened when they are fired. Agents running as critics or as batch-style analysis checks use a reporting dialogue, as shown in Figure 6 (2). This shows a list of prioritised problems with the architecture model that analysis agents have detected. The architect reviews the critic report from time to time and analysis report after they have requested agents generate one. The architect tell an agent to ignore one or more components, in which case any exception message is hidden.

A number of pre-packaged analysis agents are available for software architects to use by opening projects containing them (in the same way architects choose packaged meta-model elements and templates). Architects can also build their own analysis agents using a visual event processing language supplied by the Serendipity-II process management application [9].

An example of such an agent specification is shown in Figure 6 (3). Such agent specifications are made up of a guard, which filters architecture model change events. Each guard ultimately has a guard action which generates the exception message, recording the exception and a representation of which is presented to the user. The analysis agent may also provide one or more "fix actions", semi-automating correction of the architecture model if the architect so requests.

Serendipity-II process models can be used to control analysis agents automatically using the event filtering and actioning tool. The architect can define "coordination agents" that switch agents on/off, change their priority or the way they are fired when process model stages are enacted or finished.

6. Environment Architecture and Implementation

Figure 7 illustrates the architecture of SoftArch. SoftArch was implemented using the JComposer meta-CASE tool, which generates classes that specialise our JViews component-based architecture for multi-view, multi-user environment construction [10]. SoftArch is thus a component-based system and able to be integrated with other component-based tools by JViews facilities. SoftArch provides multiple views of software architecture models with a centralised repository and flexible view consistency mechanism. It provides a variety of collaborative work facilities, including synchronous and asynchronous editing of views, version merging and configuration management. These capabilities are similar to those of Serendipity-II and JComposer [9].

SoftArch maintains a set of meta-model projects which define the allowable components, associations, annotations and property types for a model. A set of reusable refinement templates (which are SoftArch models) allow reuse of common architectural refinements. A modelling project holds the model of the software architecture currently under development.

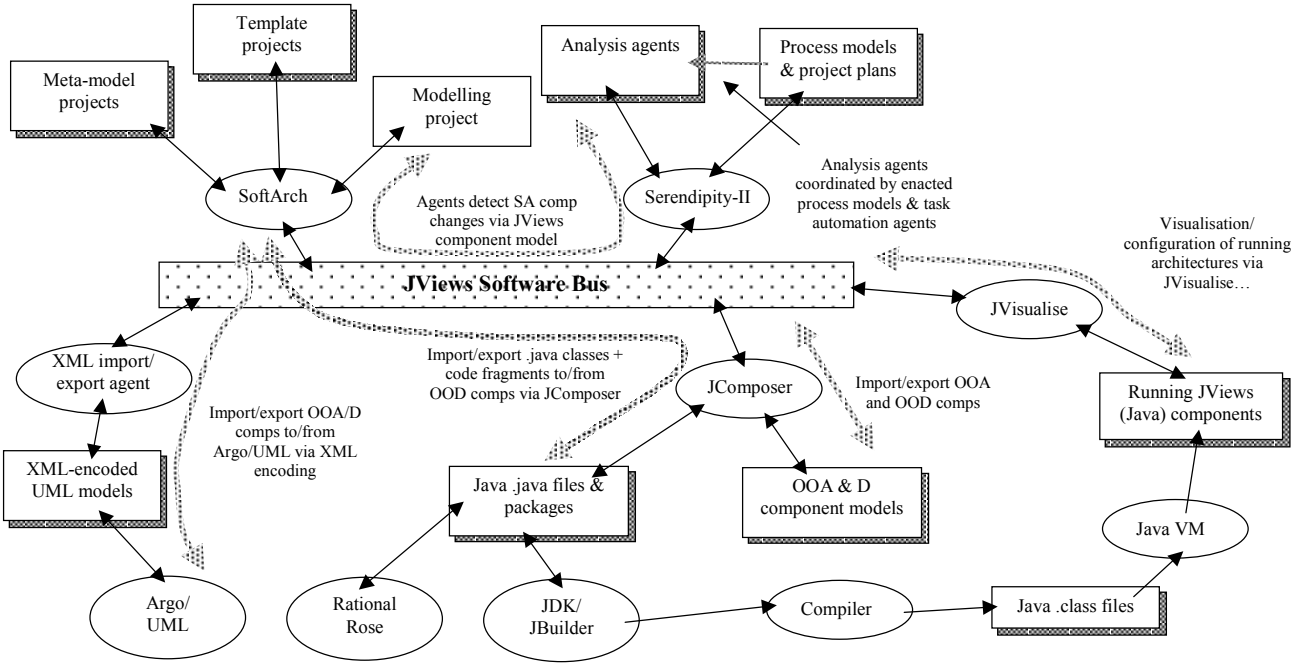

Figure 7. SoftArch architecture.

Our Serendipity-II process management environment is used to provide enactable process models and project plans to guide use of SoftArch. We also use Serendipity-II's visual task automation agent language to allow architects to build new analysis agents for SoftArch. Serendipity-II and SoftArch communicate via the JViews software bus [10]. Analysis agents in Serendipity-II monitor SoftArch component change events. Serendipity-II task automation agents can be used to co-ordinate the use of analysis agents (turning them on/off etc.), can control the meta-models being used in SoftArch, and can be used to co-ordinate work by multiple architects.

We have integrated SoftArch with JComposer, our component environment supporting OOA, design and implementation facilities, using the JViews infrastructure facilities. We have developed import and export components which import an OOA model from JComposer into SoftArch, and that can export an OOD model and code fragments from SoftArch to JComposer. JComposer generates Java source code files for these OOD-level components itself, and can reverse engineer OOD models for import into SoftArch.

We have built prototype import/export tools that use an XML to encode OOD-level components from SoftArch for import into Argo/UML, and that can transform XML-encoded Argo/UML OOA models into SoftArch. JComposer-generated classes can be used with the reverse engineering tool of Rational Rose to import SoftArch designs into Rose. Java classes generated by Rose can be reverse engineered by JComposer and then imported into SoftArch to provide a simplistic OOA-level import facility from Rose to SoftArch. JComposer-generated classes can be used in programming environments like JBuilder and JDK to complete system implementation.

We plan to annotate generated code so that our run-time component visualisation system, JVisualise [10], can be used to monitor and control running programs. The information from JVisualise will allow SoftArch visualisation tools to provide high-level visualisation of systems using SoftArch's architectural abstractions, rather than implementation-level objects. Ultimately we would like to extend this approach to allow architects and developers to use dynamic visualisations of running systems in SoftArch to modify the system structure with high-level SoftArch views, with JVisualise translating high-level manipulations into appropriate implementation-level modifications.

7. Discussion

Most existing CASE tools, such as Rational Rose [18], Argo/UML [19] and JComposer [10], provide limited abstractions for designing large system architectures. In fact, few abstractions besides OOA/D modelling and simple component and deployment diagrams are provided by most tools [11]. We have found these to be inadequate for most system development tasks from the perspective of software architecture design. In addition, most CASE tools do not adequately support refinement of OOA/D models with capture of architecture-related design rationale and linkage of components at different levels of abstraction. Few provide adequate template or reusable model support.

Component engineering tools, such as JComposer [10], JBuilder [4] and that of Wagner et al [23], provide little in the way or architecture modelling support, but focus on design- and implementation-level detail. This is necessary when developing systems, but not high-level enough for large system architecture development. Few support capture of multiple perspectives on

architecture models and different levels of abstraction and refinement relationships.

Some tools have been developed specifically for software architecture modelling or had a range of architecture modelling capabilities added. Examples include PARSE-DAT [15], ViTABaL [7], Clockworks [6], SAAMTool [12], JComposer aspects [8] and Argo/UML [19]. These typically provide limited architectural modelling support, and many are oriented to limited kinds of architectural abstractions. For example, PARSE-DAT focuses on process-oriented views of architectures, ViTABaL on tool-based abstraction and SAAMTool on structural composition. ClockWorks provides some useful, high-level architectural annotations, but these are limited to caching, concurrency and ADT replication annotations. SoftArch provides a wide, extensible range of architectural abstractions and representations, ranging from static structure and information exchange to dynamically composable systems and process synchronisation mechanisms.

Architecture Description Languages, such as Wright [1] and Rapide [16], typically focus on formal specification of architectural styles and support reasoning about the characteristics of such architectural styles. In contrast, SoftArch aims to support modelling and analysis of system architectures, with architectural components and analysis support embedded in the tool meta-models, templates and analysis agents. We have de-emphasised formal reasoning in SoftArch, although some analysis agents perform complex formal reasoning about various property values between associated components.

Few CASE tools or other environments provide adequate architecture model analysis and verification tools, and only provide limited (if any) integration and reverse engineering support. Examples include PARSE-DAT, ViTABaL, Architecture Description Languages, and ClockWorks provide some analysis support, but limited to specific kinds of domains. Argo/UML provides design critics which mainly focus on OOA and OOD-level model evaluation heuristics. Argo's critics can not be extended by users using visual language specification techniques as in SoftArch, and users have more limited control over them.

SoftArch leverages existing tool facilities, such as those of JComposer and Serendipity-II, rather than having OOA/D, code generation and process management facilities built-in. This is in contrast to tools like MetaEdit+ [13], Argo/UML [19] and Rational Rose™ [18]. These systems either provide built-in process management and code generation support or have none. They also provide rather more limited integration mechanisms via file formats, leading to less tightly integrated environments than we have with SoftArch.

8. Summary

Current approaches to software architecture modelling are not adequate for large system architecture development. SoftArch provides a new approach to modelling software architectures with an extensible meta-model of architecture abstractions, flexible and extensible visual language modelling tools, reusable refinement templates and successive refinement of architecture models. In addition, SoftArch provides user-extensible and controllable analysis agents, integrated process modelling and enactment support, and integrated OOA/D import/export and code generation facilities. These facilities are provided by the integration of SoftArch with the Serendipity-II and JComposer tools, rather than monolithic extensions to SoftArch itself.

We have used SoftArch to model the architectures of several small-to-medium distributed systems. Results of developing these systems with the aid of SoftArch have been very encouraging. We are continuing to extend and refine the SoftArch meta-model types and modelling tools as we gain experience with the environment on larger problems. We are adding new analysis tools as we find a need for them, and are building up libraries of reusable refinement templates. We are working on improved tool integration mechanisms in order to effectively use SoftArch with a wide range of 3rd party CASE tools and programming environments. We are also improving its code generation capabilities by the use of JComposer. We are planning to use annotated code to support dynamic architecture visualisation using SoftArch's high-level architectural views, and eventually to support dynamic architecture manipulation of running systems via high-level SoftArch abstractions.

References

1. Allen, R. and Garlan, D. A formal basis for architectural connection, *ACM Transactions on Software Engineering and Methodology*, July 1997.
2. Bass, L., Clements, P. and Kazman, R. Software Architecture in Practice, Addison-Wesley, 1998.
3. Booch, G., Rumbaugh, J. and Jacobson, I. The Unified Modelling Language User Guide, Addison-Wesley, 1999.
4. Borland Inc, *Borland JBuilder™*, Borland Inc, http://www.borland.com/jbuilder/, 1998.
5. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. *Pattern Oriented Software Architecture : A System of Patterns*, Wiley, 1996.
6. Graham, T.C.N., Morton, C.A. and Urnes, T. ClockWorks: Visual Programming of Component-Based Software Architectures. *Journal of Visual Languages and Computing*, Academic Press, pp. 175-196, July 1996.
7. Grundy, J.C., Hosking, J.G. ViTABaL: A Visual Language Supporting Design by Tool Abstraction, In *Proceedings of the 1995 IEEE Symposium on Visual Languages*, Darmsdart, Germany, September 1995, IEEE CS Press, pp. 53-60.
8. Grundy, J.C. Supporting aspect-oriented component-based systems engineering, In *Proceedings of 11th International Conference on Software Engineering and*

Knowledge Engineering, Kaiserslautern, Germany, June 16-19 1999, KSI Press, pp. 388-395.

9. Grundy, J.C., Hosking, J.G., Mugridge, W.B. and Apperley, M.D. An architecture for decentralised process modelling and enactment, *IEEE Internet Computing*, Vol. 2, No. 5, September/October 1998, IEEE CS Press.
10. Grundy, J.C., Mugridge, W.B., Hosking, J.G. Static and dynamic visualisation of component-based software architectures, In *Proceedings of 10th International Conference on Software Engineering and Knowledge Engineering*, San Francisco, June 18-20, 1998, KSI Press.
11. Grundy, J.C. and Hosking, J.G. Directions in modelling large-scale software architectures, In *Proceedings of the 2nd Australasian Workshop on Software Architectures*, Melbourne 23rd Nov 1999, Monash University Press, pp. 25-40.
12. Kazman, R. Tool support for architecture analysis and design, In *Proceedings of the Second International Workshop on Software Architectures*, ACM Press, 94-97.
13. Kelly, S., Lyytinen, K., and Rossi, M., "Meta Edit+: A Fully configurable Multi-User and Multi-Tool CASE Environment," In *Proceedings of CAiSE'96,* Lecture Notes in Computer Science 1080, Springer-Verlag, Heraklion, Crete, Greece, May 1996, pp. 1-21.
14. Leo, J. OO Enterprise Architecture approach using UML, In *Proceedings of the 2nd Australasian Workshop on Software Architectures*, Melbourne 23rd Nov 1999, Monash University Press, pp. 25-40.
15. Liu, A. Dynamic Distributed Software Architecture Design with PARSE-DAT, In *Proceedings of the 1998 Australasian Workshop on Software Architectures*, Melbourne, Australia, Nov 24, Monash University Press.
16. Luckham, D.C., Augustin, L.M., Kenney, J.J., Veera, J., Bryan, D. and Mann, W. Specification and analysis of system architecture using Rapide, IEEE Transactions on Software Engineering, vol. 21, no. 4, April 1995, 336-355.
17. Mowbray, T.J., Ruh, W.A. *Inside Corba: Distributed Object Standards and Applications*, Addison-Wesley, 1997.
18. Quatrani, T. Visual Modeling With Rational Rose and Uml, Addison-Wesley, 1998.
19. Robbins, J. Hilbert, D.M. and Redmiles, D.F. Extending design environments to software architecture design, *Automated Software Engineering*, vol. 5, No. 3, July 1998, 261-390.
20. Sessions, R. *COM and DCOM: Microsoft's vision for distributed objects*, John Wiley & Sons 1998.
21. Shaw, M. and Garlan, D. Software Architecture : Perspectives on an Emerging Discipline, Prentice Hall, 1996.
22. Urnes, T. and Graham, T.C.N. Flexibly Mapping Synchronous Groupware Architectures to Distributed Implementations. In *Proceedings of Design, Specification and Verification of Interactive Systems (DSV-IS'99),* 1999.
23. Wagner, B., Sluijmers, I., Eichelberg, D., and Ackerman, P., Black-box Reuse within Frameworks Based on Visual Programming, In *Proeedings of the. 1st Component Users Conference,* Munich, July 1996, SIGS Books, pp. 57-66.