Generation of Distributed System Test-beds from High-level Software Architecture Descriptions

John Grundy[1], Yuhong Cai[1] and Anna Liu[2]

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

[2]Software Architectures and Component Technologies
CSIRO Mathematical and Information Sciences,
Locked Bag 17, North Ryde, NSW 1670, Sydney
Australia
Anna.Liu@cmis.csiro.au

Abstract

Most distributed system specifications have performance benchmark requirements. However, determining the likely performance of complex distributed system architectures during development is very challenging. We describe a system where software architects sketch an outline of their proposed system architecture at a high level of abstraction, including indicating client requests, server services, and choosing particular kinds of middleware and database technologies. A fully-working implementation of this system is then automatically generated, allowing multiple clients and servers to be run. Performance tests are then automatically run for this generated code and results are displayed back in the original high-level architectural diagrams. Architects may change performance parameters and architecture characteristics, comparing multiple test run results to determine the most suitable abstractions to refine to detailed designs for actual system implementation. We demonstrate the utility of this approach and the accuracy of our generated performance test-beds for validating architectural choices during early stages of system development.

1. Introduction

Most system development now requires the use of complex distributed system architectures and middleware [1, 22]. Architectures may use simple 2-tier clients and a centralised database; 3-tier client, application server and database; multi-tier, decentralised web, application and database server layers; and peer-to-peer communications [1, 19, 22]. Middleware may include socket (text and binary protocols), Remote Procedure (RPC) and Remote Method Invocation (RMI), DCOM and CORBA, HTTP and WAP, and XML-encoded data [6, 16, 17]. Data management may include relational or OO databases, persistent objects, XML storage, files. Integrated solutions combining several of these approaches, such as J2EE and .NET, are also increasingly common [18].

Typically system architects have stringent performance (and other) quality requirements their designs must meet. However, it is very difficult for system architects to determine appropriate architecture organisation, middleware and data management choices that will meet these requirements during architecture design [15, 9]. Architects often make such decisions based on their prior knowledge and experience. Various approaches exist to validating these architectural design decisions, such as architecture-based simulation and modelling [3, 14, 23], performance prototypes [12, 6, 11], and performance monitoring and visualisation of similar, existing systems [3, 21]. However, simulation tends to be rather inaccurate, performance prototypes require considerable effort to build and evolve, and existing system performance monitoring requires close similarity and, very often, considerable modification effort to gain useful results.

We describe SoftArch/MTE, an integrated tool allowing software architects to accurately test the performance of their architecture designs using high-level architecture design diagrams. Architects sketch high-level system descriptions, including client, server, database and host elements, and expected client requests and server and database services. SoftArch/MTE automatically generates a fully functional, multiple client and server deployable performance test-bed. This incorporates code using the specified middleware and data management approaches and adheres to the specified architecture organisation. This performance test bed is deployed and run on multiple client and server hosts. Generated code and deployment annotations automatically capture performance measurements and relay them to SoftArch/MTE. Performance data is then displayed in the high-level architecture diagrams and by using external data visualisation tools. Results from different test runs, architecture organisations and middleware and data management choices can be compared and recorded for future reference.

We first motivate this work with a simple distributed system development example, then overview the key elements of our SoftArch/MTE approach. We illustrate our currently-supported architecture meta-model elements in SoftArch/MTE, along with an example high-level architecture design. We describe and illustrate our

extensible code generation approach using XML and XSLT transformations, which generate complex performance test bed code. We describe and illustrate test bed deployment, performance testing and result visualisation in SoftArch/MTE. We compare and contrast our research with related work and evaluate its strengths and weaknesses, concluding with a summary of our main research contributions and directions for future work.

2. Background

Consider the development of a system to support an on-line video store library [8], supporting customer on-line video search/reservation and staff in-store video rental management tasks. Example interfaces for such a system are illustrated in Figure 1. Video store staff might use desktop or browser-based applications that connect to application and/or web servers or even direct to database(s). Customers interact with clients (applets or HTML) that connect to web servers and/or application servers, in turn connecting to other servers and one or more databases e.g. holding staff, customer, video, video rental etc details. Data processing may be centralised in single server objects or spread across clients or multiple servers. Middleware connectivity may use HTML, Java RMI, DCOM, CORBA, XML and so on. Application server objects maybe COM, CORBA or Enterprise Java Beans. Data management may be in relational, object or XML databases, or some even in files.

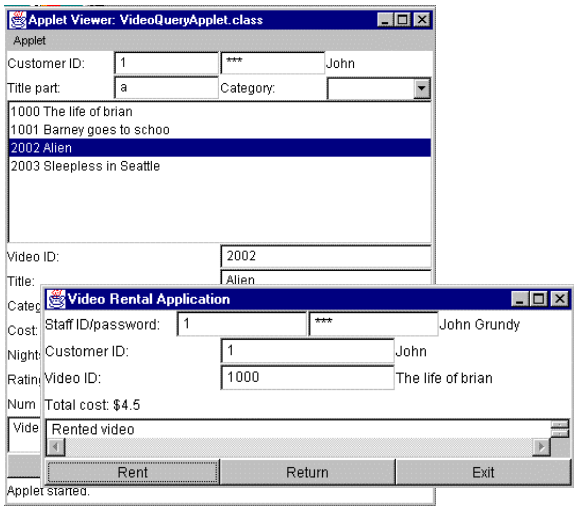

Figure 1. Parts of a simple on-line video system.

The software architect would typically have some performance criteria any chosen architecture design must meet e.g. maximum number of users, response time for different user requests and data processing services, etc. They may also have some hardware and possibly software constraints e.g. must run on Windows/LINUX machines; must run on low-end desktop machine; must run over 56kbps modem connection; must use either CORBA or DCOM protocol, and so on.

As part of the Middleware Technology Evaluation Project (MTE), we have done extensive performance evaluation of basic architecture models using a wide range of middleware and data management technologies for such systems, including Enterprise Java Beans, DCOM, MQ Series, TIBCO, and Java Messaging Service [7, 6]. We have built many performance "test beds" – simple distributed system implementations designed to extensively benchmark performance of basic distributed system architectural approaches and related middleware technologies. While the results of these evaluations give developers great assistance in identifying the general behavioural characteristics of basic software architectures using different middleware, developers must still extensively develop prototypes of their own particular architectural and middleware choices to get an accurate understanding of their architectures' likely performance.

We have also developed a tool, SoftArch, for designing complex software architectures, generating partial object-oriented designs and visualising developed system architecture performance [8]. The following sections describe our work unifying this MTE performance work and SoftArch modelling/visualisation work to provide an environment for automated distributed system architecture design performance analysis. SoftArch/MTE generates MTE-style performance test beds from high-level SoftArch architecture design diagrams, automatically runs multiple tests and captures performance measures, and visualises these results back in SoftArch diagrams. The aim of this work is to fully-automate test bed generation, deployment and results analysis for software architects from high level system descriptions, but ensure they receive very accurate estimates of the eventual, fully-developed system performance. A rapid, exploratory architecture design process is supported, resulting in much reduced architecture validation time and improved eventual architecture quality.

3. Outline of SoftArch/MTE Process

SoftArch/MTE supports evolutionary architecture modelling, test bed generation, performance analysis and revision. Figure 2 outlines the way SoftArch/MTE is used by software architects. Steps 2-6 are fully automated.

The architect first constructs a high-level architecture design, specifying clients, servers, remote server objects and database tables, client-server, server-server, client/server-database requests and server services, and various kinds of connectors between these architectural abstractions (belongs-to, runs-on, network connection, etc) (1).b They also specify various properties: client, server and database host machine; number and frequency of requests (e.g. 1000 times; continuous; every 0.25 seconds; etc); database table and request complexity (e.g. one row select; 100 row select/update; one row insert/delete etc); middleware protocol (e.g. CORBA using Visibroker 4.0; TCP/IP socket using textual XML document; etc); and so on. Available modelling elements and their peoperies are specified in an extensible SoftArch meta-model. SoftArch diagrams and architecture element properites may be versioned, copied to/from reusable templates,

collaboratively viewed and edited with other users, and so on [8].

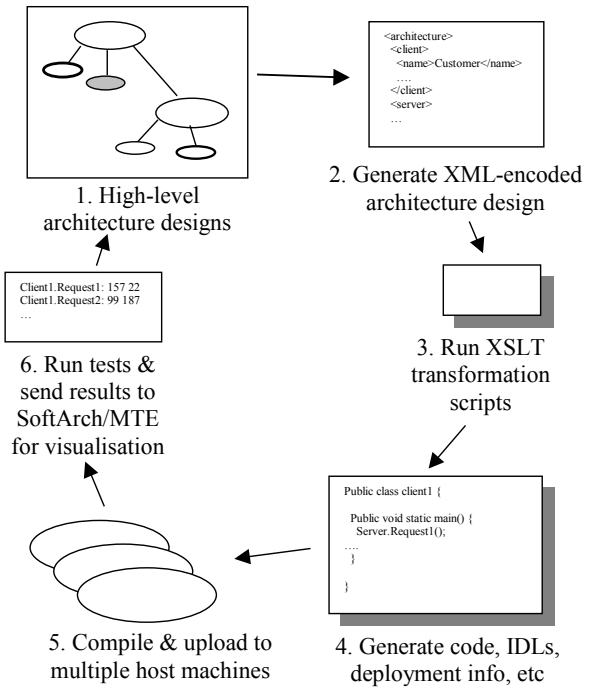

Figure 2. SoftArch/MTE architecture analysis process.

The architect instructs SoftArch/MTE to generate an XML encoding of the architecture model (2). This is then passed through a number of XSLT (XML style sheet transformations) scripts (3), which generate Java, C++, Delphi etc code, along with CORBA and COM IDL files, EJB deployment descriptors, database table creation and population scripts, compilation and start-up scripts, and so on (4). This generated code is fully-functional and is compiled without architects viewing or editing it in any way (5).

Compiled (deployable) client and server program code is then up-loaded to the specified client, server and database host machines by sending them to a deployment agent running on those machines via Java RMI. The generated client and server programs and appropriate database servers are started on all hosts and clients wait for a SoftArch/MTE signal (via their deployment agent), or a scheduled start time, to begin execution i.e. sending requests to servers. SoftArch/MTE client and server code annotations are usually used to capture performance measures, though we can also generate scripts to configure performance monitoring programs to collect performance measures. Once tests complete, deployment agents collect results (usually from client and server program output files) and send these to SoftArch via RMI (6).

SoftArch annotates architecture diagrams in various ways to show performance measures, or invokes a data visualisation tool to show performance details and summary charts (we use MS Excel™). Multiple test run results e.g. using different middleware, databases and client/server request mixes can be visualised together. Architecture designs and their performance results can be versioned and compared by architects. 3rd party tools e.g. MS Excel™ can be invoked by SoftArch/MTE to provide richer data visualisations. Results can be saved to SoftArch repositories for long-term recording and reuse.

4. Modelling Software Architectures

Figure 3 shows SoftArch being used to model a candidate design for the video system architecture [8]. SoftArch provides a variety of predominantly graphical architecture modeling tools (1) and an extensible meta-model of available architecture elements, connectors and properties. It also provides a set of "design critics" (2) that monitor software architecture model changes and give unobtrusive user feedback. Data collected by performance monitoring annotations in code developed from SoftArch models is used to visualize running system performance at high-levels of abstraction using SoftArch diagrams (3).

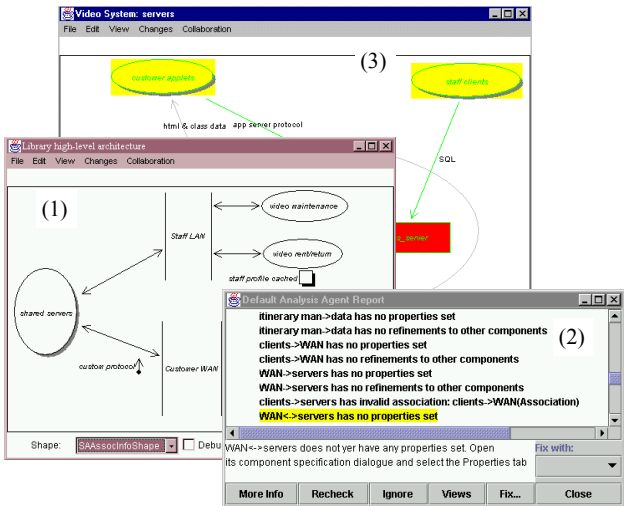

Figure 3. Example of SoftArch in use.

We have developed a SoftArch meta-model to support encoding high-level, complex software architecture designs enabling generation of MTE-style test bed code. Part of this meta-model is shown in Figure 4 (1). Abstractions include clients, servers, server objects, database tables, various requests, server services (basically a set of server-side requests, for multi-tier architectures), various kinds of connectors, and properties for each abstraction. Constraints specify valid connections and property values.

Architects design their distributed system architectures using these meta-model abstractions and SoftArch's visual modelling tools. Figure 4 (2) shows an example 3-tier architecture for part of the on-line video system. Staff and customer clients have a number of requests e.g. find video/customer/rental, add/update rental item, update customer details etc. Customer clients connect to a set of remote objects (VideoManager, CustomerManager etc), encapsulated in two server processes.

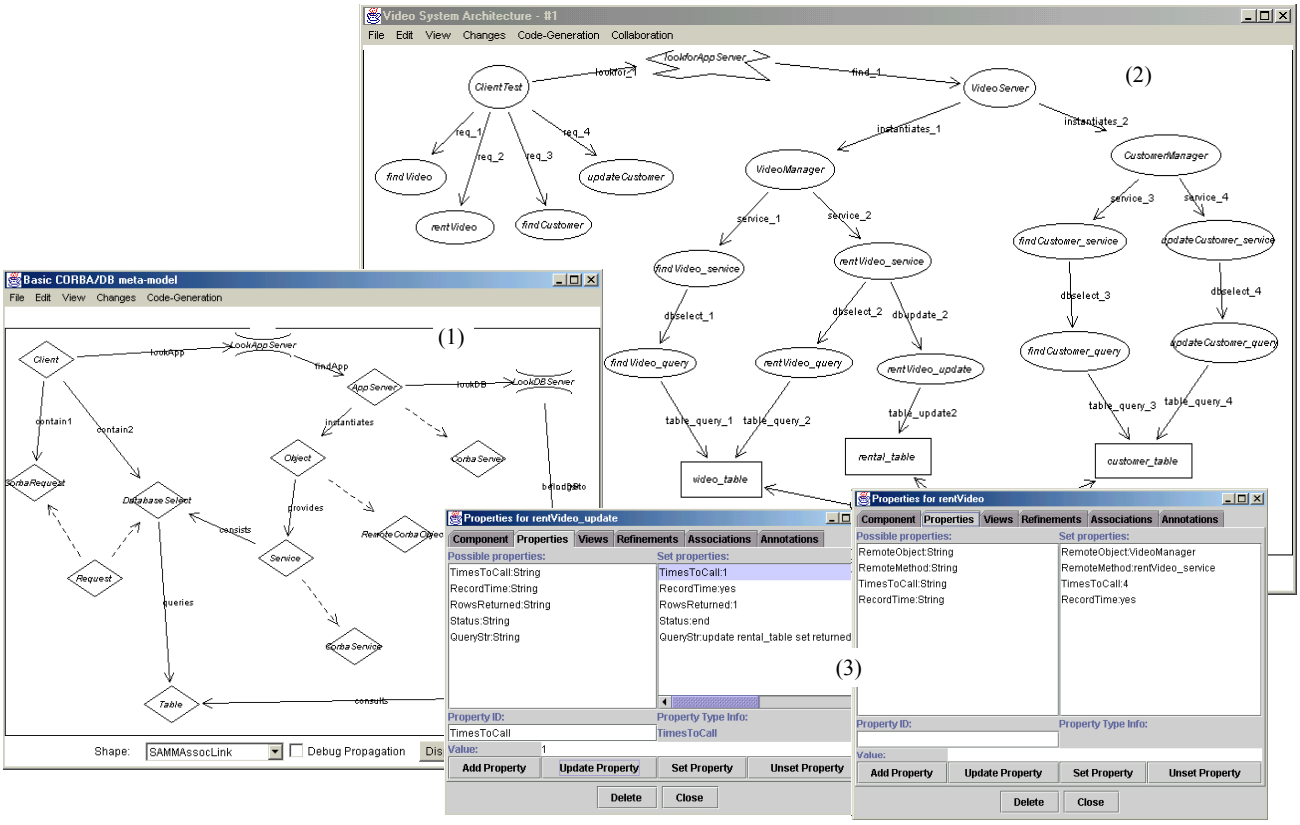

Figure 4. SoftArch/MTE meta-model abstractions and example high-level distributed system architecture.

These could be EJB objects/servers, CORBA or COM objects, CORBA server or DCOM server processes etc (we use remote CORBA objects as an example in this paper). A database stores data (customers, staff, videos, rentals, etc). Connections specify request/service ownership, client-server-server connectivity and so on. Various properties of architecture elements and connectors are specified in dialogues (Figure 4 (3)). These include number and kind of each request expected; kind of remote service, remote service requests, database table properties (expected number of rows/columns), client and server process hosts, and so on. For complex systems, various over-lapping and sub-views can be used by architects to manage complexity. Automated view consistency management is provided by SoftArch [8].

5. Generating Performance Test-bed Code

From high-level architecture designs as illustrated in the previous section, SoftArch/MTE can generate test beds that provide very accurate performance measures for a developed system (well – as accurate as the mix of client/server requests, architecture connectivity, database table complexity, etc the architect is prepared to specify). The more detailed the architecture design the more accurate the performance measure results, though our experiments with SoftArch/MTE and comparison to completed projects' performance have shown useful (and quite close to actual developed system) performance estimates can be achieved from even only 10-15 minutes of high-level architecture design.

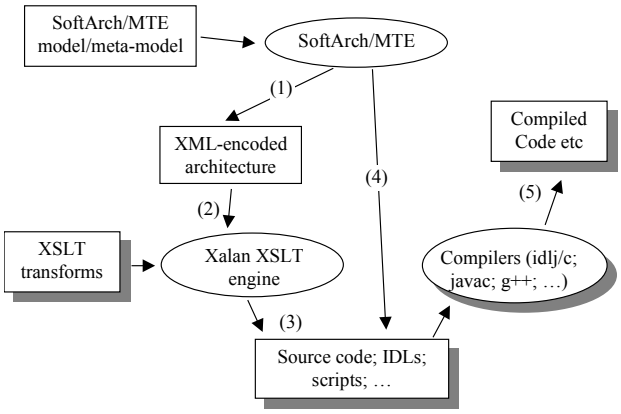

Figure 5. Code generation process.

The code generation process used by SoftArch/MTE is outlined in Figure 5. SoftArch/MTE traverses the architecture design using element/connector types and meta-model data to generate a full XML encoding of the design (1). A set of XSLT transformation scripts and an XSLT engine (2) "transform" various parts of the XML into program source code, IDLs, deployment descriptors, compilation scripts, deployment scripts, database table construction and population scripts, and so on (3). Client and server program code is compiled automatically by SoftArch/MTE using generated compilation scripts (4) to produce fully-functioning, deployable test-bed code (5).

We chose to use an XML encoding and a set of XSLT transformation scripts to do our code generation for several key reasons:

- Code generation and architecture modelling are de-coupled from each other
- The SoftArch/MTE models are easily translated into XML, readable by a range of other applications
- The XSLT transformations are easily modified and extended, or new scripts for new kinds of source code, compilation script, IDL etc generation added, without the need for complex coding or modifying SoftArch/MTE source or its generated XML data
- New meta-model abstractions can be added which add new XML items without breaking the existing code generation
- The XML encoding is potentially usable for other purposes e.g. import into a CASE tool for full system design/implementation.
- Other architecture design tools in the future might be used to generate the XML architecture encoding but use our code generation XSLT scripts.

Figure 6 (1, 2) shows part of the on-line video system architecture from Figure 4 encoded in XML. Figure 6 (3) shows part of an XSLT transformation script used to convert parts of the XML matching CORBA client requests into CORBA client code in Java, and Figure 6 (4) shows some of resultant generated client code. Each of our XSLT scripts use "templates" to match parts of the XML-encoded software architecture descriptions (a). Each template transforms part of the XML data into program code, IDL definition, compilation and deployment script parts, and so on (b). This is done by specifying static output data (e.g. fixed code and script fragments) and dynamic output data (c) (e.g. copying XML-encoded source data values such as names, numbers etc into the output) in the XSLT scripts. Different XSLT transforms can match the same XML-encoded architecture data, generating different code (e.g. corba-client.xsl generates server object look-up functions for encoded CORBA server objects, whereas corba-server.xsl generates object creation and registration code for this same XML-encoded data). These XSLT transformation scripts can be straightforwardly modified or new scripts added without requiring any SoftArch code or XML encoding modification. Currently SoftArch applies all available scripts to an XML encoded architecture, even if some transformations aren't relevant for that architecture (they simply produce no generated code).

6. Testing and Visualising Performance

Figure 7 outlines the code performance testing process. Generated code is compiled by SoftArch, using generated compilation scripts (1). The compiled code/IDLs/descriptors etc and scripts to deploy/run them on a host are up-loaded to remote client and server hosts using remote SoftArch/MTE deployment agents (2).

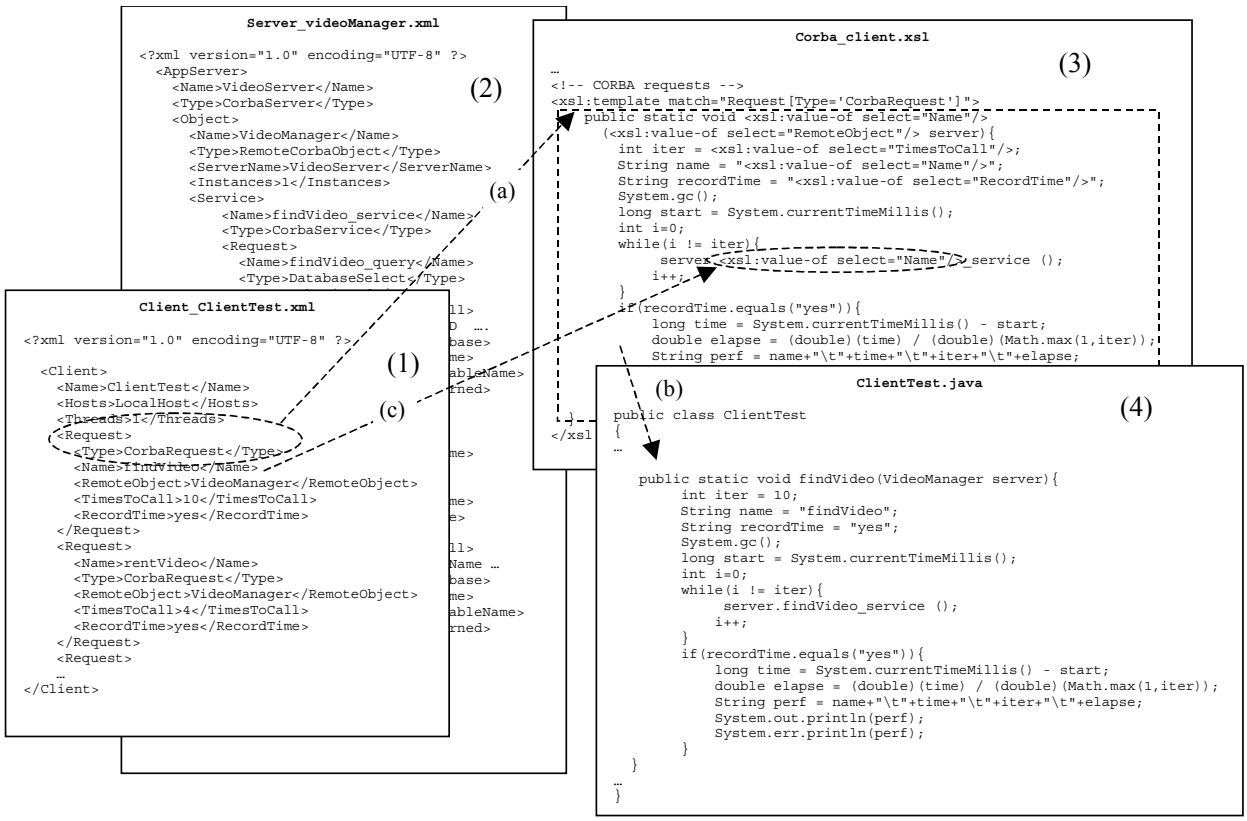

Figure 6. Generated SoftArch/MTE XML architecture encoding, example XSLT transform and generated code.

The client and server programs are then run: server programs started; EJB components deployed into EJB servers; database servers started and database table initialisation scripts run; and then clients started (3). Clients look up their servers and then await SoftArch sending a signal (via their deployment agent) to run, or may start execution at a specified time.

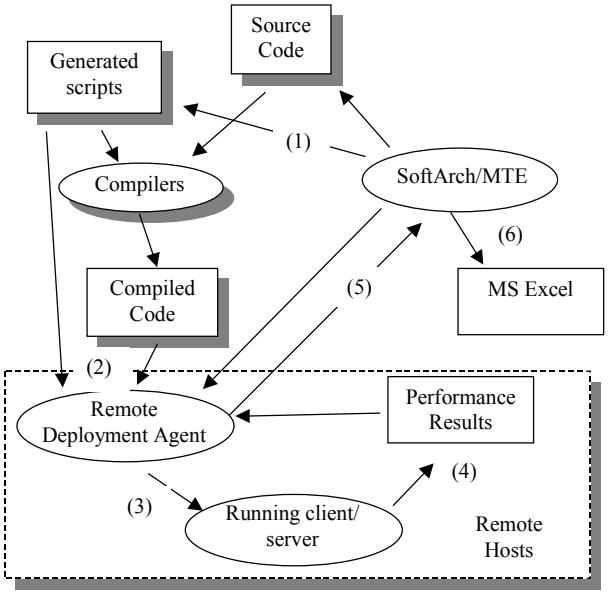

Figure 7. System deployment and test run process.

Clients run their server requests, typically logging performance timing results for different requests to a file (4). Servers do like-wise. 3rd party performance measuring tools can also be deployed to capture performance information, and are configured by SoftArch/MTE-generated scripts. Performance results are sent back to SoftArch/MTE for visualisation (5), possibly using 3rd party tools like MS Excel™ (6).

We built a basic deployment agent to allow SoftArch/MTE-generated programs, components and databases to be automatically deployed for architects on multiple hosts. These also play a role in co-ordinating performance test initiation and results transmission to SoftArch/MTE. We used Java RMI to upload code, deployment scripts and database initialisation scripts to client and server host deployment agents, and to send performance measures back to SoftArch/MTE.

Performance results currently include name of request/service, name of owning object/process, number of times called, overall time taken, and data stored/retrieved. SoftArch/MTE records these performance results against appropriate architecture elements, summarising results across multiple test bed client and server instantiations. Summarised results include number of calls made by a request/to a service; average and total time to complete request/service; average and total time spent in a request/service; and average and total database accesses/updates performed. Results are visualised by annotating SoftArch/MTE diagrams (e.g. via shading and line thickness), as shown in Figure 8, and in dialogues showing total values.

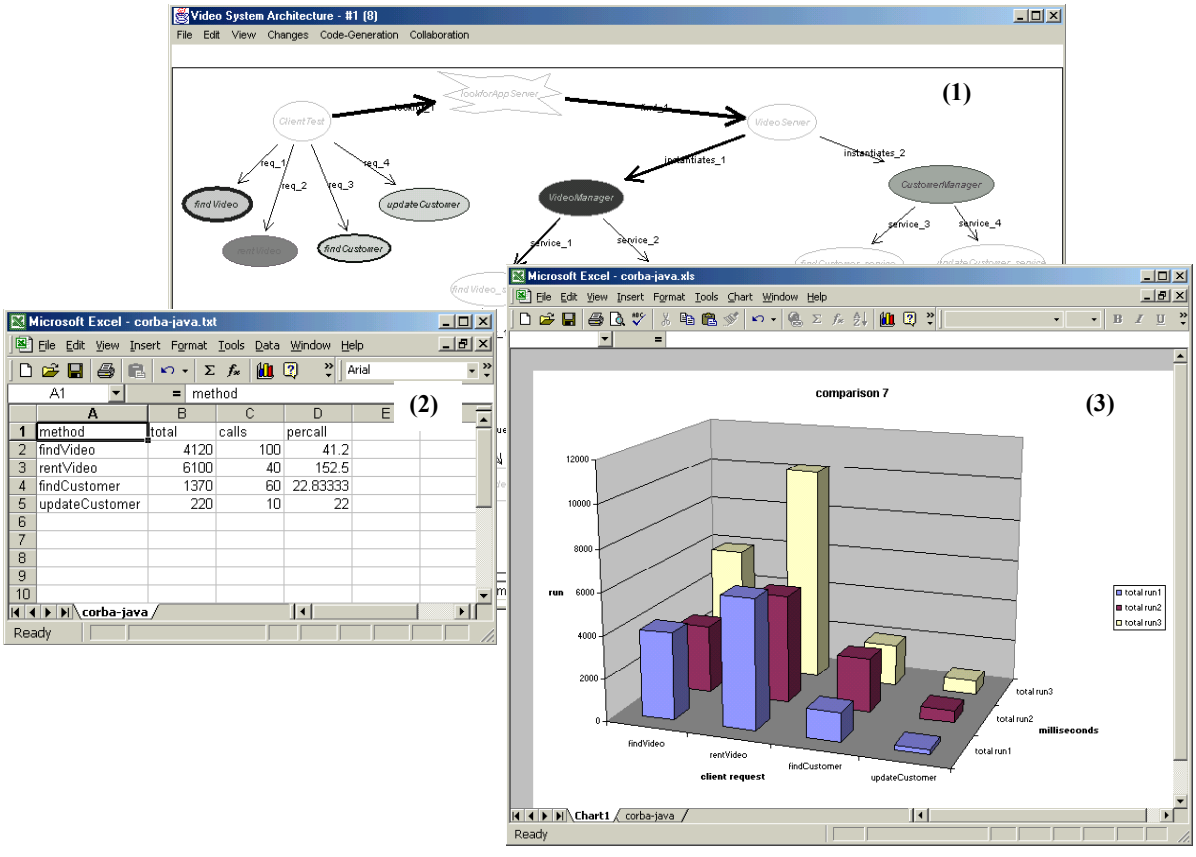

Figure 8. Visualising test run results in SoftArch/MTE and MS Excel™.

The user requests the particular performance results they wish to view. Annotations are used to visually indicate light vs heavy requesting/loading, small vs large time consumption, small vs large data transferral, etc. Figure 8 (1) shows the video system diagram from Figure 4 annotated after a performance test (showing total time taken summarised data). Figure 8 (2) shows performance result values for one client collected by SoftArch/MTE and opened for display using a MS Excel™ spreadsheet. Figure 8 (3) shows details of several performance tests of this system with different architectures (multiple CORBA objects in one server process; remote data manager objects split across 2 server processes; and data managers across 3 server processes). These results were appended into one file and a reusable MS Excel™ chart used to visualise these values.

7. Discussion

The need to evaluate software architecture and distributed systems middleware performance has been recognised for a long time [11, 15, 9]. Developers typically build prototype systems i.e. performance test beds by hand to test architecture performance [6, 11]. This is very time-consuming and if even limited architecture changes are made, often whole new prototypes needed to be built to validate the new architecture. SoftArch/MTE generates test bed prototypes whose performance measures are generally very accurate but require a tiny fraction of the developer effort that hand-built prototyping approaches need.

Various architecture and middleware performance simulation and modelling methods and tools have been developed [5, 16, 14, 23]. These make it easier for architects to express and explore likely architecture performance, but their performance results are often quite inaccurate. Specification of architectures and visualisation of simulation-derived performance measures are often predominantly text-based, lacking our SoftArch/MTE's easy to use, high-level, graphical abstractions.

A variety of middleware, network, database and software performance monitoring and architecture visualisation tools exist [15, 2, 4, 10, 21]. These all typically require a fully developed system in order to be used, meaning either prototyping a system by hand or analysing a built system with a related architecture to that planned. Visualisations in these tools are often quite low-level i.e. tend to focus on programmatic features rather than high-level architectural abstractions.

Benchmarks published for various middleware, architecture and database systems can be useful for architects to gauge likely relative performance for different design choices [20, 6]. Unfortunately most system architectures are a complex mix of design choices (architecture layout and divisions of responsibility; middleware and database choices; and host machine and network characteristics). Accurate performance measures thus can only be gained from a fully-developed system, or from a prototype sufficiently close to the eventual system in generated code and example processing requests and data.

We have used SoftArch/MTE to design, generate and performance evaluate a range of simple architectural designs like those of the video system with differing middleware and database choices. We have also used it to design and evaluate a number of quite complex system architectures (including architectures for an on-line, co-operative travel planning system, a large, complex enterprise business system, and an integrated health informatics system). We have compared SoftArch/MTE's performance analysis results we obtained for these three systems to those of fully developed distributed systems for these applications (two developed by ourselves, one developed by a local software company). We performance tested different SoftArch/MTE architecture and middleware choices, and compared these results to those obtained manually performance testing modified versions of these three medium-sized software systems. We found all SoftArch/MTE performance results extremely close (within 5-10%), and in one instance almost identical, to the real systems' performance measures, despite very little effort required to sketch and modify the high-level architecture designs in SoftArch/MTE (for all these examples less than 30 minutes each).

While very little software architect time needs to be spent to obtain these highly accurate results (compared to days of prototyping for even simple architectures), the approach has its limitations. The architect is constrained to use the provided meta-model abstractions and XSLT code generation scripts. If they wish to evaluate other architecture abstractions or different middleware, they or others must extend the meta-model and code generation support. It is challenging for an architect to estimate the likely mix of client requests and server-to-server requests in the eventual system, and thus inaccurate loading will produce inaccurate performance measures. However, all testing approaches (prototypes, simulation and even monitoring fully developed systems) suffer from the same problem when using estimated loading rather than real users and data. We have noticed the ease in specifying discrete client and server requests in SoftArch/MTE architecture designs does tend to lead architects to specifying overly-simple designs when a real system would actually have a much more complex mix of client-server, server-server and server-database requests. If only a small number of client and server hosts are available for testing, SoftArch/MTE can not directly estimate likely performance on larger numbers of machines as performance simulation techniques may do.

We are adding new code generation scripts to SoftArch/MTE and meta-model abstractions to support generating message-oriented middleware (e.g. MQ Series and Tuxedo) code and DCOM component requests, and are looking to generate code for HTTP and .NET-based middleware. We are building Wizards to allow architects to specify full client and server requests and services for complex architecture designs without needing to draw individual element icons and connectors and specify their properties. We are working on improved, richer

performance result visualisations, including showing results for different test runs on (slightly) modified architectures together in the same diagram. We are investigating using a "standard" XML-based architecture design encoding, like xADL [13], to allow other architecture design tools to use our code generation and deployment scripts.

8. Summary

SoftArch/MTE provides a high-level, extensible architectural modelling language allowing architects to quickly design and evolve key architectural characteristics. These specifications are rich enough to allow fully-functional test bed code to be generated, deployed and run, all without any developer involvement. A set of extensible XSLT transformations scripts are used to transform XML-encoded architecture designs into test bed client and server program code and compilation/deployment scripts, with compiled systems automatically uploaded and configured to multiple host machines. Test runs are performed on these client and server host machines and results automatically captured, aggregated and visualised by SoftArch/MTE in the original high-level architecture diagrams. SoftArch/MTE architecture designs and performance figures can be versioned and stored for future comparison of different architecture designs and their performance test results. Experiences to date with SoftArch/MTE have demonstrated it provides a useful, accurate automated architecture performance analysis environment for complex distributed systems development.

References

1. Bass, L., Clements, P. and Kazman, R. Software Architecture in Practice, Addison-Wesley, 1998.
2. Beaumont, M. and Jackson, D. Visualising Complex Control Flow. In *1998 IEEE Symposium on Visual Languages*, Halifax, Canada, September 1998, IEEE CS Press.
3. Chen, M., Tang, M. and Wang, W. Software Architecture Analysis - A Case Study, In *Proceedings of COMPSAC'99*.
4. Egyed, A. and Kruchten, P., Rose/Architect: a tool to visualize architecture, In *Proceedings of the 32nd Hawaii International Conference on System Sciences,* January 1999, IEEE CS Press.
5. Gomaa, H., Menascé, D., and Kerschberg, L. A Software Architectural Design Method for Large-Scale Distributed Information Systems, *Distributed Systems Engineering Journal*, Sept. 1996, IEE/BCS.
6. Gorton, I. And Liu, A. Evaluating Enterprise Java Bean Technology, In *Proceedings of Software - Methods and Tools*, Wollongong, Australia, Nov 6-9 2000, IEEE CS Press.
7. Gorton, I., Liu, A., et.al. Evaluating Enterprise Middleware Technologies, *Middleware Technology Evaluation Report Series*, available from CSIRO Publishing www.cmis.csiro.au/adsat/mte.htm and Cutter Consortium, www.cutter.com/itgroup/reports, February 2001
8. Grundy, J.C. and Hosking, J.G. High-level Static and Dynamic Visualisation of Software Architectures, In *Proceedings of 2000 IEEE Symposium on Visual Languages*, IEEE CS Press.
9. Grundy, J.C. and Liu, A. Directions in Engineering Non-Functional Requirement Compliant Middleware Applications, In *Proceedings of the 3rd Australasian Workshop on Software and Systems Architectures*, Sydney, Australia, Nov 2000, Monash University.
10. Hill, T., Noble, J. Visualizing Implicit Structure in Java Object Graphs, In *Proceedings of SoftVis'99*, Sydney, Australia, Dec 5-6 1999
11. Hu L., Gorton, I. A performance prototyping approach to designing concurrent software architectures, In *Proceedings of the 2nd International Workshop on Software Engineering for Parallel and Distributed Systems*, IEEE CS Press, pp. 270 – 276.
12. Jurie, M.R., Rozman, I., Nash, S. Java 2 distributed object middleware performance analysis and optimization, *SIGPLAN Notices*, vol.35, no.8, Aug. 2000, ACM Press, pp.31-40.
13. Khare, R., Guntersdorfer, M., Oreizy, P., Medvivovic, N. and Taylor, R.N. xADL: Enabling Architecture-Centric Tool Integration With XML, in *Proceedings of the 34th Hawaii International Conference on System Sciences*, Jan 3-6 2001, Maui, Hawaii, IEEE CS Press.
14. Liu, A. Dynamic Distributed Software Architecture Design with PARSE-DAT, In *Proceedings of Software – Methods and Tools,* Wollongong, Australia, Nov. 2000, IEEE CS Press.
15. McCann, J.A., Manning, K.J. Tool to evaluate performance in distributed heterogeneous processing. In *Proceedings of the Sixth Euromicro Workshop on Parallel and Distributed Processing*, IEEE CS Press, 1998, pp.180-185.
16. Petriu, D., Amer, H., Majumdar, S., Abdull-Fatah, I. Using analytic models for predicting middleware performance. In *Proceedings of the Second International Workshop on Software and Performance*, ACM 2000, pp.189-94.
17. Petrovski, A. and Grundy, J.C. Web-enabling an integrated health informatics system, In *Proceedings of the 7th International Conference on Object-oriented Information Systems*, Calgary, Canada, August 27-29 2001, Springer LNCS.
18. Shannon, B., *Java 2 platform, enterprise edition : platform and component specifications*, Addison-Wesley, 2000.
19. Shaw, M. and Garlan, D. *Software Architecture*, Prentice Hall, 1996.
20. TechMetrix, The Application Server Directory, www.techmetrix.com/trendmarkers/techmetrixasd.php3.
21. Topol, B., Stasko, J. and Sunderam, V., PVaniM: A Tool for Visualization in Network Computing Environments, *Concurrency: Practice & Experience*, Vol. 10, No. 14, 1998, pp. 1197-1222.
22. Vogal, A. CORBA and Enterprise Java Beans-based Electronic Commerce, *International Workshop on Component-based Electronic Commerce*, Fisher Center for Management & Information Technology, UC Berkeley, 25th July, 1998.
23. Woodside, C. Software Resource Architecture and Performance Evaluation of Software Architectures, In *Proceedings of the 34th Hawaii International Conference on System Sciences*, IEEE CS Press, Maui, HA., Jan 2001.