Grundy, J.C., Cai, Y. and Liu, A. SoftArch/MTE: Generating Distributed System Test-beds from High-level Software Architecture Descriptions, Automated Software Engineering, Kluwer Academic Publishers, vol. 12, no. 1, January 2005, pp. 5-39

SoftArch/MTE: Generating Distributed System Test-beds from High-level Software Architecture Descriptions

John Grundy[1, 2], Yuhong Cai[2] and Anna Liu[3]

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

[3]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, for example the number of particular kinds of transactions per second required to be supported by the system. However, determining the likely eventual performance of complex distributed system architectures during their development is very challenging. We describe SoftArch/MTE, a software tool that allows software architects to sketch an outline of their proposed system architecture at a high level of abstraction. These descriptions include client requests, servers, server objects and object services, database servers and tables, and particular choices of middleware and database technologies. A fully-working implementation of this system is then automatically generated from this high-level architectural description. This implementation is deployed on multiple client and server machines and performance tests are then automatically run for this generated code. Performance test results are recorded, sent back to the SoftArch/MTE environment and are then displayed to the architect using graphs or by annotating 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. Further tests may be run on refined architecture descriptions at any stage during system development. We demonstrate the utility of our approach and prototype tool, and the accuracy of our generated performance test-beds, for validating architectural choices during early system development.

1. Introduction

Most system development now requires the use of complex distributed system architectures and middleware [1, 35]. Architectures are quite varied and systems may use simple 2-tier clients with a centralised database; 3-tier client, application server and database divisions of responsibility; multi-tier, decentralised web, application and database server layers; and peer-to-peer communications [1, 33, 35]. Using each of these approaches an architect has a wide range of choice as to where to locate data processing (e.g. client-side or server-side), whether to cache data (e.g. in an application sever to reduce database overheads), and whether to use combinations of architectural styles for different parts of the same overall system. Middleware used to connect distributed components of a system 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 [10, 27]. 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 becoming increasingly common [32].

Typically system architects need to work within quite stringent performance (and other) quality requirements that systems implemented using their designs must eventually meet. For example, a system might need to be able to handle a specified maximum number of concurrent users with a particular user request satisfied in a maximum specified time. However, it is very difficult for system architects to determine appropriate architecture organisation, middleware and data

management choices that will meet such requirements during architecture design [27, 13]. Very often architects make such decisions based on their prior knowledge and experience. In order to enable developers to get a more accurate assessment of architectural design decisions various approaches have been developed to validate architectural designs. These include architecture-based simulation and modelling [4, 24, 36], performance prototypes [19, 10, 17], and performance monitoring and visualisation of similar, existing systems [4, 34]. Simulation of architecture and middleware performance tends by its very nature to be more or less inaccurate, giving results of limited usefulness. Performance prototypes may give much more accurate estimates of implemented system performance but require considerable development effort to build and any evolution of architecture design means new or substantially modified prototypes need to be developed. Analysing existing system performance through monitoring requires close similarity between the existing system and proposed system and often considerable modification may be required to gain any useful performance results.

We have developed SoftArch/MTE, an integrated tool allowing software architects to accurately test the performance of their architecture designs by generating performance test-beds from 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 using a visual Architecture Description Language. From these descriptions SoftArch/MTE automatically generates a ruable, multiple client and server deployable performance test-bed. This test bed code incorporates dummy code to generate client, server and database requests using the specified middleware and data management approaches and adheres to the specified architectural organisation. Our generated performance test beds are deployed and run on multiple client and server hosts, providing a realistic deployment scenario for the code. Generated code and deployed application annotations automatically capture performance measurements as the test bed code runs. Recorded performance data is then relayed to SoftArch/MTE where the data is displayed in the high-level architecture diagrams and by using external data visualisation tools such as MS Excel™. Results from different test runs, architecture organisations and middleware and data management choices can be compared.

We first motivate our research with a simple distributed system development example, an on-line video system, back grounding the two previous research projects we have combined to produce the SoftArch/MTE tool. We critique related work into architecture and middleware performance evaluation techniques and then give an overview of the key elements of our SoftArch/MTE approach. We illustrate how an architect constructs a high-level architecture description using our currently-supported architecture meta-model elements. We describe and illustrate our extensible code generation approach that use XML and XSLT transformations to generate complex performance test bed code. We describe and illustrate the test bed deployment, performance testing and result visualisation process we have developed. We conclude with an analysis of the utility of SoftArch/MTE for performance analysis of software architecture designs and with a summary of our main research contributions and directions for future work.

2. Motivation

All distributed systems have a variety of non-functional constraints that an implementation of the system should adhere to, for example security, data integrity, reliability, performance, exception handling, user interface approach and so on [2, 5]. One of the driving factors in choosing a particular architectural organisation and communications middleware for a system is to ensure these constraints can be met by a design and implementation derived from this architecture [10, 2, 33]. One of the most difficult constraints to ensure is met during architecture (and system) design is performance. This is due to a number of factors: insufficient knowledge of detailed design decisions; unknown performance characteristics of many possible middleware and database choices; unknown performance characteristics of new and often existing software on particular hardware and network platforms; and the complexity of the design meaning combinations of architecture, design and implementation choices may never have been tried by the development team before [2, 10]. This makes validating architectural design decisions in terms of them meeting required system performance requirements virtually impossible during early design.

Consider the development of a system to support an on-line video store library [12], supporting customer on-line video search/reservation and staff in-store video rental management tasks. Some example interfaces for such a system are illustrated in Figure 1 (a). Just one of the many possible architectures that could be used to implement this system is shown in Figure 1 (b). In this example, video store staff use desktop applications connecting to the database(s). Customers interact with user interfaces that connect to application servers, in turn connecting to possibly other servers and one or more databases e.g. holding staff, customer, video, video rental etc details. Data processing may be centralised or spread across clients or servers. Middleware may be HTML, Java RMI, DCOM, CORBA, or XML. Server objects may be COM, CORBA or Enterprise Java Beans. Data management may use relational, object or XML databases, or files.

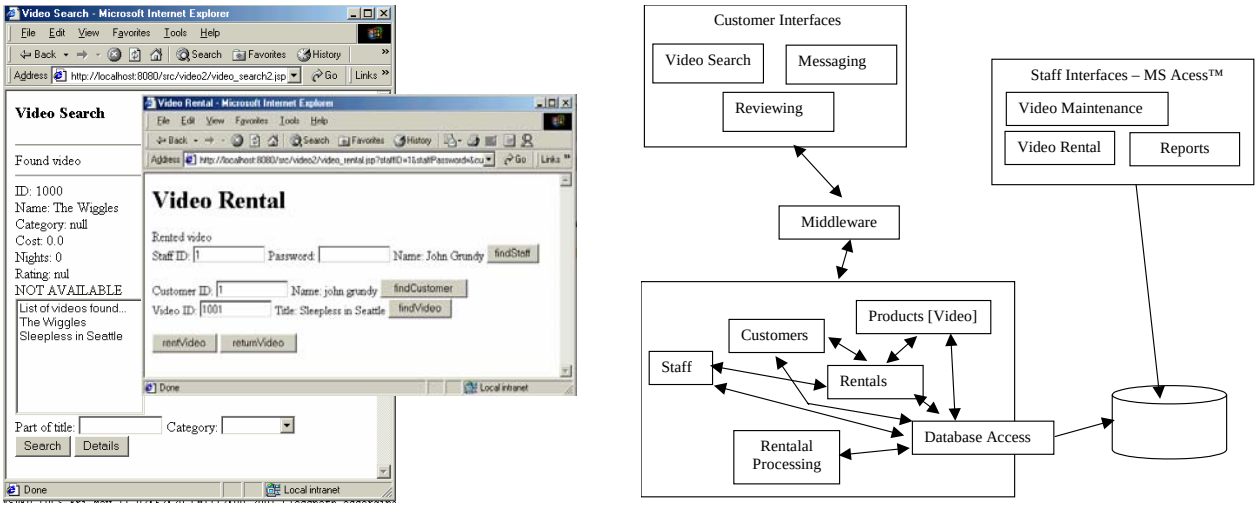


Figure 1. Parts of a simple on-line video system.

The software architect typically has some performance constraints from a non-functional analysis for the system that any chosen architecture design must meet. For example, such constraints might include maximum number of users, response time for different user requests and data processing services, and so on. There may possibly be some further constraints that the architect needs to work within, for example some hardware and 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 middleware protocol, must use SQL Server™ 7 database, and so on. In addition, if a system under design must interact with existing systems as part of their operation then the performance and other constraints of the existing applications and their deployment will impact the performance of parts of the new system.

To determine a suitable architecture to use for a system, and appropriate middleware and database choices, an architect typically relies on past experience. They often use rapid prototyping or simulation tools to verify that design choices will likely result in architectural non-functional constraints being met (e.g. performance, number of concurrent users, reliability and so on). Previously developed benchmarks for particular architectures and middleware choices can also assist in guiding design. In this paper we describe an approach to generating performance test-beds (essentially rapid prototypes) directly from a software architecture description. The aim is to enable architects to more quickly and iteratively understand the performance impacts of their architecture-level design decisions. This paper provides a more detailed description of the work previously reported in [15], and includes a number of extensions to the architecture modeling, code generation and evaluation results.

3. Related Work

The need to evaluate software architecture and distributed systems middleware performance has been long recognised by researchers and practitioners [17, 27, 13, 7]. The main approaches to assessing architecture and middleware performance have been the development of test beds (prototypes), the use of simulation and other modelling techniques to estimate likely performance of architectures, and the use of performance monitoring technologies to assess performance of developed systems. The first two approaches are usually used to try and understand likely performance of an architecture before constructing a full system or substantial part of a system based on it. The last approach may be used either before designing a new system similar to an existing one being monitored or may be used after having developed a system and then needing to gain better understanding of aspects of its architecture in order to improve its performance.

Developers typically build prototype performance test beds by hand [10, 17]. This approach is usually very time-consuming for all but the most simple of architectures [10]. If middleware selection is changed or if even simple re-organisation of the architecture design is made very often a whole new test bed prototype needs to be built to assess the likely performance of the new architecture. If when a prototype was first developed if failed to capture adequately the kinds of client/server operations or division of responsibilities that end up being embodied in an evolving architecture design, repeated modifications or replacement of the prototype are necessary. Given that most performance prototypes are in many ways fairly mechanical to produce [17] it would be desirable to generate such code from higher level models of software architectures with the generated test bed code able to be easily regenerated when these models are changed.

Benchmarks published for various middleware and database systems can be useful for architects to gauge possible relative performance for different design choices [7, 10]. These are usually obtained from common benchmarking programs – basically performance test-bed prototypes developed for general use so results of technologies and vendor solutions can be directly compared. However most real 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 for a particular architecture design thus can only be gained from a prototype sufficiently close to the intended eventual system.

In order to avoid having to build continuously evolving performance test bed prototypes a number of architecture and middleware performance simulation and modelling methods and tools have been developed [1, 9, 7, 18, 27, 24, 29, 36]. These make it easier for architects to express and explore likely architecture performance by providing higher-level models of architectures and using these to produce simulated or computed performance estimates for the models. All of these approaches can only provide estimates of possible architecture model performance, however, as they factor out a variety of performance-impacting attributes e.g. some host computer and network characteristics, performance of particular versions of application software and so on [26]. In addition, the accuracy of the performance estimates are highly dependent on the details of the model used and the ability of the simulation and modelling techniques used [18, 27, 22, 36]. This leads to performance results derived from abstract architecture model analysis possibly being quite inaccurate. In addition, the specification of architecture characteristics and the visualisation of simulation-derived performance measures are often predominantly text-based. These approaches provide software architects with low-level detail about architecture performance estimates requiring further analysis and visualisation.

The performance of deployed distributed systems has been of interest to developers for many years and has resulted in the development of a wide variety of middleware, network, database and software performance monitoring and architecture visualisation tools [27, 3, 8, 16, 34]. Many of these approaches aim to provide high-level abstractions for viewing architecture structure and/or performance results [8, 16]. While Visualisations in many of these tools are often quite low-level i.e. tend to focus on low-level programmatic features rather than high-level architectural abstractions, some monitoring tools capture low-level performance information and aggregate this and present it to the user in a higher-level abstraction [34, 27]. Most however do not present performance results in a form compatible with the same visual abstractions used when modelling architecture designs making interpretation of results more difficult. Most performance monitoring techniques require a fully developed system in order to be used. During architecture design the only way to get such systems to evaluate are to build prototypes or try and evaluate similar systems to the one under development. The former approach means much effort particularly if the design continues to evolve and the later is only useful if the two systems are very similar in likely transaction mix and structural organisation.

Many researchers have investigated ways of visualising software architecture structure and behaviour. These range from static architecture modelling approaches usually focusing on structure and to a lesser degree behaviour [31, 20, 23, 25] to dynamic architecture visualisation and metrics analysis [8, 36, 24, 11]. We chose to use our SoftArch architecture modelling and analysis tool in this research as it provides a mix of both static (structure and behaviour modelling) and dynamic (performance and other architecture metric visualisation) support facilities within one integrated environment [12].

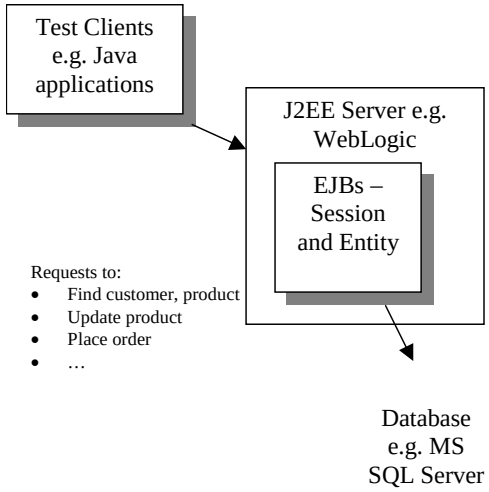
4. Background: MTE and SoftArch

The Middleware Technology Evaluation (MTE) project at the CSIRO [5] has been working for several years on developing benchmarks for a wide variety of software architectures and associated middleware and database technologies. An additional aim of this work has been to better understand the impact of various middleware approaches and implementation technologies on system performance [10]. This work has included assessing the performance of systems built using Enterprise Java Beans, DCOM, MQ Series, and Java Messaging Service, and assessing the relative performance of different vendor solutions using each of these technologies. The project team has built many performance "test beds" that have been used to gather performance metrics for a wide variety of middleware technologies and vendor solutions. These test beds are simple distributed system implementations designed to extensively benchmark performance of using basic distributed system architectural approaches implemented with particular middleware technologies.

Figure 2 (a) shows an example MTE test-bed architecture used to evaluate the performance of basic J2EE-based enterprise applications. A simple order processing application was built using J2EE technology and some performance testing clients and server instrumentation code used to gather performance results [5, 10]. Different J2EE server and database implementations can be tried out with the test-bed to measure their relative performance. Multiple client

instances on different host machines can be used or a single client with multiple threads. Other MTE test-beds use different architecture choices e.g. J2EE Bean-managed vs. Container-managed persistence, and make use of different middleware technologies e.g. CORBA, MQ Series and .NET.

Figure 2 (b) shows some simplified pseudo-code examples for example clients and servers used in the J2EE order processing test-bed. The client(s) runs one or more requests on the server EJBs and measure their performance. The clients can be quite complex, mixing a range of different server requests and reporting their performance. The MTE clients can also be configured to run requests on the server with no pause, to measure total raw transaction throughput obtainable, or with pausing between server requests to simulate user interactions with clients.



Client:

1. look up server objects (EJBs)
2. create required number of client threads
3. get initial time
4. start threads
5. in each thread do n times:
    a. findCustomer() or updateProduct() or placeOrder()
    b. optional pause before next request
6. get final time
7. report results

Server:

PlaceOrder():

1. begin transaction
2. select customer from database
3. select product from database
4. insert order into database
5. commit transactions

Figure 2. (a) Example of test-bed architecture; and (b) some partial test-bed code examples.

The results of these test-bed performance evaluations provide developers assistance in identifying the general performance characteristics of a range of basic software architectures using different middleware and database technology choices. However, a major problem of this approach is that in order to assess each middleware approach/vendor solution for different architecture designs, even quite simple ones, new prototype test beds must be developed. This is very time-consuming. In addition, these test beds only assess the deployment of the middleware solutions in simple, exemplar architectures, such as the simple on-line order processing example above. Developers must still extensively develop prototypes with additional details about their planned system architecture in order to get an accurate understanding of this architecture's likely performance.

In a different project we developed a tool, SoftArch, for designing complex software architectures, generating partial object-oriented designs and visualising fully-implemented system architecture performance [12]. This tool allows software architects to model high-level architecture designs using a visual language and refine their designs using successive levels of abstraction into design-level classes. Figure 3 shows SoftArch being used to model a candidate architecture design for the video system in Figure 1 [12]. SoftArch provides a variety of predominantly graphical architecture modelling 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), showing the user relative time consumption by architecture components as well as loading on network connections, object creation metrics and so on.

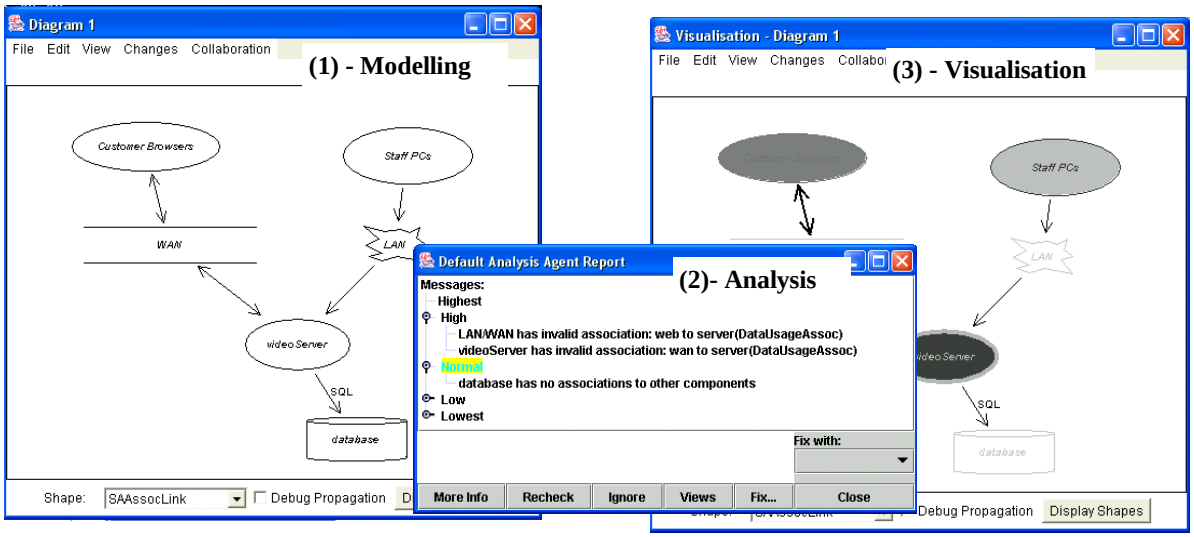


Figure 3. Example of SoftArch in use.

We wanted to try and unify our MTE work and our SoftArch work to see if we could take SoftArch architecture specifications, which include key architectural abstractions, middleware choices, client and server deployment information and so on, and generate MTE-style performance test-beds directly from these high-level architectural models. From the various MTE test-beds that had been developed [5], we recognized that almost all of these test beds could indeed be automatically generated and run from SoftArch models. To achieve this we took SoftArch's saved architecture models and used them as the input for a code generator producing MTE-style performance test bed code and batch control scripts. Our new SoftArch/MTE environment generates MTE-style performance test beds from 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 was to fully-automate MTE-style performance test bed generation, deployment of generated code and scripts, and capture and presentation of performance results for software architects. This environment supports a rapid, exploratory architecture design process where architects can easily make changes to architectural design and middleware choices and from these high level system descriptions receive accurate estimates of the eventual, fully-developed system performance. This approach aims to much reduce architecture validation time and improve eventual developed system architecture quality from a performance perspective.

5. The SoftArch/MTE Performance Evaluation Process

Performance results obtained from standard benchmarking suites and MTE-style prototypes provide accurate estimates of architecture performance as they run using real code on real machines. Similarly analysis of the performance of real systems via monitoring obtains accurate results as to architectural component behaviour when a system embodying an architecture design is deployed. However both of these approaches suffer from the effort to build performance prototypes or are only useful after a system has been implemented. In contrast the use of performance simulation of high-level models of architectures allows architects to easily specify and evolve the architecture specification but at the cost of having to accept often very imprecise simulation results due the huge complexity in modelling all performance-impacting factors on a real system's performance. SoftArch/MTE adopts the high-level specification and results-presentation approach of simulation-based approaches. However it automatically generates real executable performance test bed code that is deployed on real client and server machines and is monitored to obtain accurate real-system performance results.

The SoftArch/MTE prototype environment supports integrated, evolutionary architecture modelling, test bed generation, performance analysis and evolution. Figure 4 outlines the process whereby software architects obtain performance results from real code generated by SoftArch/MTE. 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). 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 properties are specified in an extensible SoftArch/MTE meta-model.

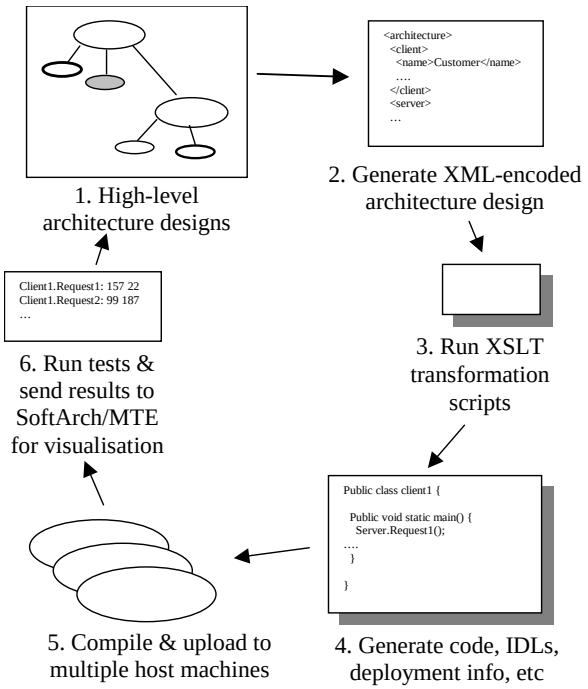


Figure 4. SoftArch/MTE architecture analysis process.

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 a fully working performance test-bed, compiled without architects viewing or editing it (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 highlight the performance measures captured from running the generated test beds. SoftArch/MTE can also generate performance summary analysis and invoke a 3rd party data visualisation tool to show performance details and summary charts (we use MS Excel™ to do this). Multiple test run results using different middleware, databases and client/server request mixes can be visualised together. Architecture designs and their performance results can also be versioned within SoftArch/MTE and the performance results of these different architecture design options compared by architects.

6. High-level Software Architecture Modelling

SoftArch is comprised of a meta-model defining all available software architecture modeling abstractions (types), a model containing software architecture designs, system requirements (constraining architecture design information) and software designs (design-level classes refined from architecture abstractions), and a set of “design critics” that automatically assess designs and provide unobtrusive feedback to users. In addition a visual meta-model editor and visual model editing tools provide (mostly graphical) viewing and editing support [12].

We have developed a SoftArch meta-model to support encoding high-level, complex software architecture designs for the purpose of supporting the generation of MTE-style test bed code. The main elements of this meta-model are shown in our visual meta-model editor in Figure 5. Details of the element types and properties are shown in The key architectural modelling abstractions we have defined for SoftArch/MTE include clients, servers, server objects, database servers, and database tables. Each client or server object may have one or more requests to another object or to a database. A server object provides one or more remote services, basically a grouped set of server-side requests, for multi-tier architectures. These client element requests and server element services may be used at the architectural level to represent a number of grouped implementation-level methods. An architectural element may represent a high level abstraction e.g. "Clients" or a low-level one e.g. "RemoteVideoDataManager". Architecture elements are linked by connectors of different kinds e.g. belongs-to, implements, data usage, method call, message passing, event subscribe/notify, hosted-by.

A number of properties for each of these architectural abstractions exist. These may be structural properties e.g. names, types, roles, arities and constraints over structure. These are denoted in Figure 6 with "AP". Other properties are used by the performance test-bed code generator e.g. number of times to call a server request, number of concurrent threads to create for a client or server, pause duration (if any) between making requests, number of rows and columns a database table has, number of rows expected to be returned or updated by a database query, number of arguments a server request expects and so on. These are denoted by "CG" in Figure 6. Properties may have single values, multiple values or expressions. Structural properties express constraints over the architectural model and both structural and code generation properties are used by the SoftArch/MTE code generator in the production of test-beds code and scripts. Architectural abstractions may be successively refined by "refinement relationships" [12] e.g. a high-level "Servers" abstraction to multiple lower-level abstractions e.g. VideoDataManager, RentalDataManager, DatabaseServer and so on. Properties of these abstractions are also successively refined and consistency checking between supported [12].

A user of SoftArch/MTE is able to model architectures using these available abstractions which are understood by our test bed code generator. The architect parameterises software architecture designs with available element and connector properties. For example, they may specify a method call between client request and server service is implemented by CORBA and 100 calls are to be made, and have code generated to implement this in their test bed. They may then change this to RMI and 250 calls for generation of a new test bed and subsequent test run. The available abstractions can be changed or added to using the visual meta-model editor (e.g. to define a more specialised kind of architectural element or provide further characteristics about an element) but the test bed code generator scripts need to be modified if this is done.

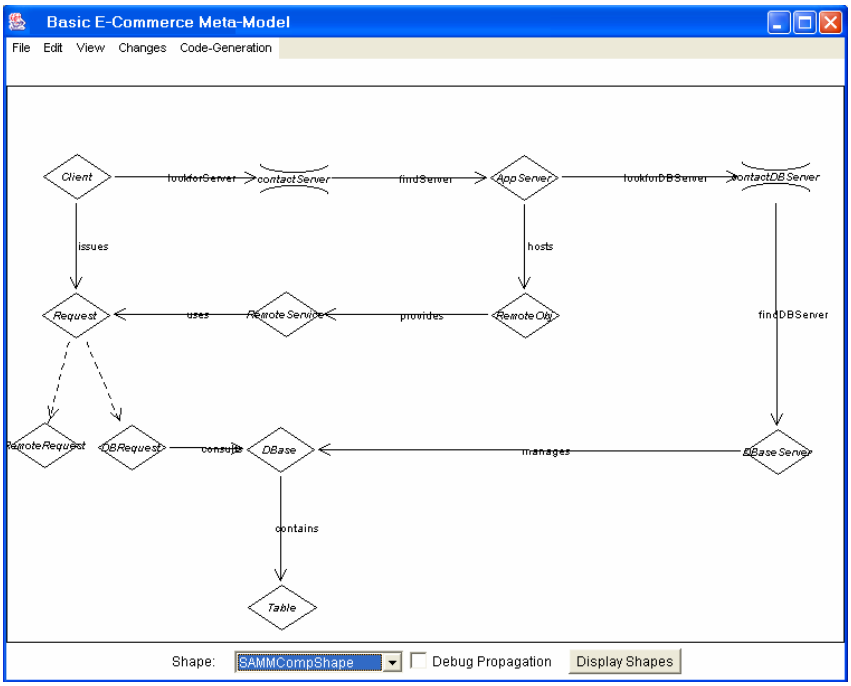


Figure 5. A visual representation of the SoftArch/MTE meta-model for E-commerce applications.

Element Type	Main Attributes	Property Description
Client	ClientType (AP, CG) Threads(CG)	Type of a client. For example, a client can be a browser, a CORBA client, or a RMI client. Number of clients that will be running during performance testing.
RemoteRequest	RemoteServer (AP, CG) RemoteObject(AP, CG) RemoteMethod(AP, CG) RecordTime(CG) TimesToCall(CG)	The name of remote server, to which the remote request is launched The name of remote object, which provides services required by the remote request The name of remote service, which completes the remote request A switch/boolean that defines if the performance testing results are recorded or not. Repetition time of certain operations during performance testing
DBRequest	QueryType (AP, CG) Dbase(AP, CG) Table(AP, CG) TimesToCall(CG) RowsReturned(CG) RecordTime(CG) Caching	Type of query for test bed and performance testing, such as 'select', 'update', 'insert', etc. The name of queried database The name of queried table Repetition times of certain operations during performance testing Number of returned results of the database request A switch/boolean that defines if the performance testing results are recorded or not. A switch/boolean that defines if query results are cached or not
AppServer	RemoteObjs (AP, CG) Type (AP, CG)	Names of all objects this application server hosts Type of the application server, such as CORBA, RMI, and J2EE……
RemoteObject	Type (AP, CG)	Type of the remote object. Type could be CORBA, RMI, and EntityBean……
RemoteService	Arguments (CG) Threading (CG) ConcurrencyControl (CG) RecordingTime (CG)	Arguments used by the service/operation A Boolean that records if this service symbols multi-threads character or single-thread character A Boolean parameter that records if the service has concurrency control or not A switch/boolean that defines if the performance testing results are recorded or not.
DBServer	Dbases (AP) ServerType (AP, CG)	Name of all databases this database server hosts Type of database server, such as MSSQL, Cloudscape, and ORACLE……
DBase	Name (AP, CG) Type (AP, CG)	Database name Type of database server, such as MSSQL, Cloudscape, and ORACLE……
Table	Name (AP, CG) PrimaryKey (CG) Rows (CG) Columns (CG)	Table name The primary key of the table Number of rows of data expected in table Number of columns expected in table

Figure 6. Some SoftArch/MTE meta-model types and properties.

Figure 7 shows an example 3-tier architecture for part of the on-line video system modelled using the SoftArch/MTE meta-model abstractions from Figure 5. In this example, staff and customer client programs have a number of requests that they can make on remote services hosted elsewhere e.g. find video/customer/rental, add/update rental item, update customer details etc. Some of these requests are simple (one remote call), others involve several remote requests. For each client, a number of "users" is specified and for each client request a number of times to call the remote service(s) and time to pause (if any) between invocations can be specified by the architect. This information configures the actual server loading tests that will be run by SoftArch/MTE's generated performance test-beds.

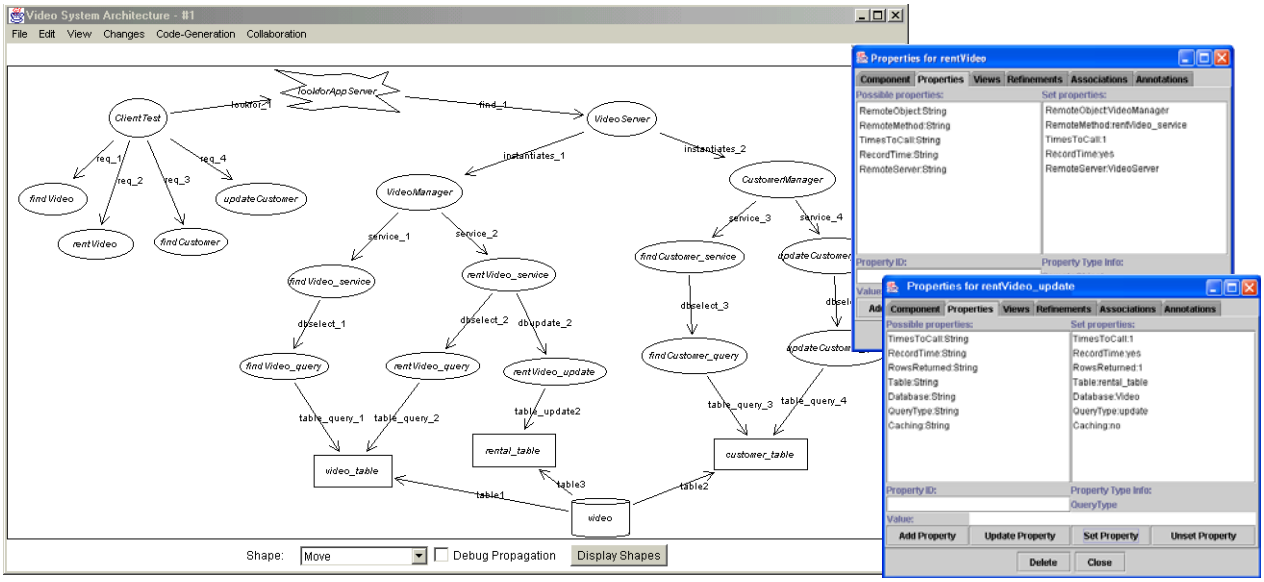


Figure 7. SoftArch/MTE meta-model abstractions and example high-level distributed system architecture.

In this example the customer clients connect to a set of remote objects (VideoManager, CustomerManager etc). These could be EJB objects/servers, CORBA or COM objects, CORBA server or DCOM server processes, Java Server Pages (JSPs), servlets or ASPs and so on. We use remote CORBA objects in most of our examples in this paper. In this architecture design, two remote objects are hosted by a single CORBA application Similarly, for this architecture design a single database stores data (customers, staff, videos, rentals, etc). Connections between architectural elements specify request/service ownership, client-server-server connectivity and so on.

Various properties of architecture elements and connectors are specified via dialogues. 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. Multiple views can be created by architects to enable specification of complex architectures or to provide various ways of viewing the same architecture design to help manage complexity. Figure 8 shows some of the properties for some elements defined for this architecture. Elements with a '*' have their properties shown at the right. Four different clients have been defined whose server requests will be run concurrently by the performance test bed. A single server defines two remote server objects. Each server object provides some remote services that are invoked by the clients across a CORBA middleware infrastructure. In this architecture (simple 3-tier), the server-side services run database transactions to select, insert and update data.

Meta-Type	Element	Sample Attributes&Values (for Element marked '*')
Client	ClientTest *	ClientType: CORBA Threads: 15
RemoteRequest	findVideo rentVideo * findCustomer updateCustomer	RemoteServer :VideoServer RemoteObject: VideoManager RemoteMethod: rentVideo_service RecordTime: yes TimesToCall: 1
DBRequest	findVideo_query rentVideo_query rentVideo_update * findCustomer_query updateCustomer_query	QueryType :update Dbase: video Table: rental_table TimesToCall: 1 RowsReturned: 1 RecordTime: yes Caching: no
AppServer	VideoServer *	RemoteObjs: VideoManager, CustomerManager Type:CORBA
RemoteObj	VideoManager * CustomerManager	Type: CORBA

RemoteService	findVideo_service rentVideo_service * findCustomer_service updateCustomer_service	Arguments: 1000 Threading: false ConcurrencyControl: true RecordingTime: yes
Dbase	Video *	Type:MSSQL
Table	Video_table * Rental_table Customer_table	PrimaryKey: ID Rows: 2500 Columns: 12

Figure 8. Some element types and their property values for the simple 3-tier video system architecture.

This video system could be realised by a wide variety of different architectures and middleware and database technologies to that shown in Figure 7. Each different architecture and middleware choice has advantages and disadvantages. The architecture from Figure 7 provides a single point of failure and the danger of bottlenecking the remote object services and database server. In addition, different clients, client requests and server services may suit a different architectural arrangement. For example, simple high-volume read requests may be cached or split across multiple machines with mirrored databases, whereas different kinds of transaction processing updates are centralised on one or more machines according to their data access or remote object service access patterns [1, 10, 35]. Architects may want to use a different set of architectural design choices and compare their estimated performance metrics.

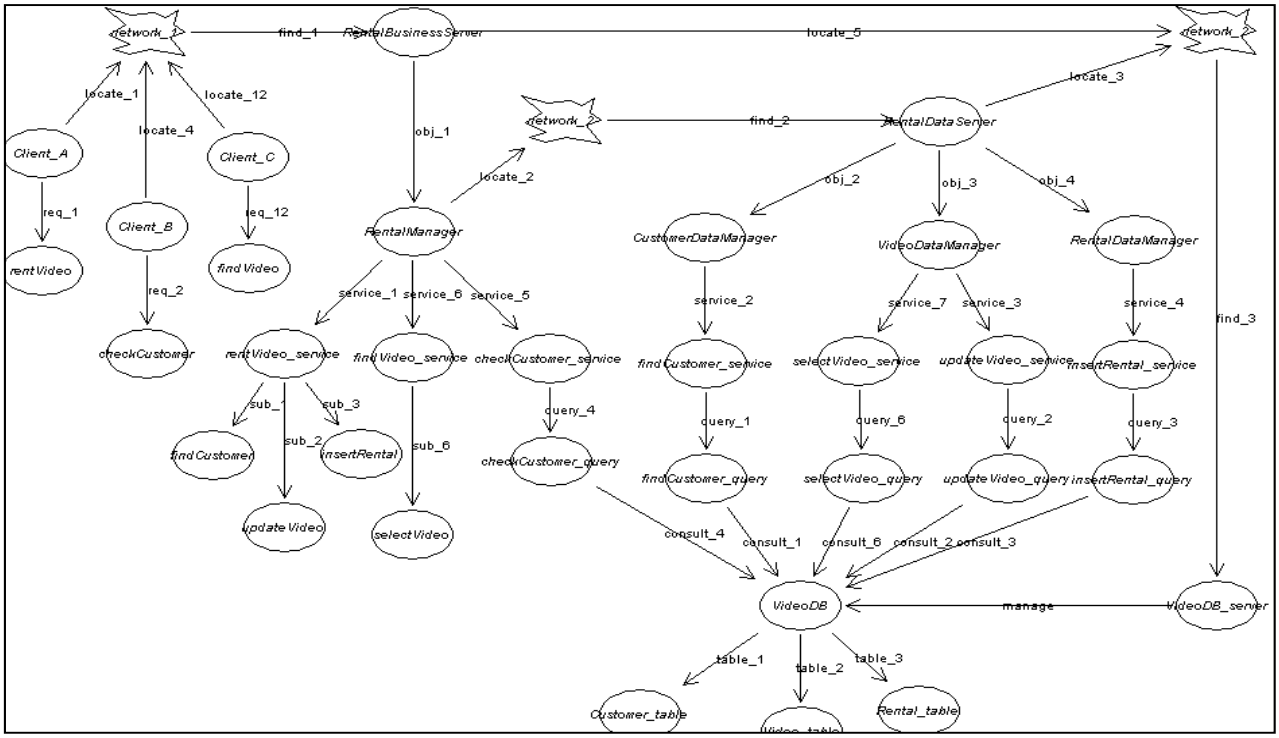


Figure 9. An alternative video system architecture design.

An alternative architecture design for our video system modelled in SoftArch/MTE is shown in Figure 9. In this design the architect has used a combination 3-tier and 4-tier architecture. Two CORBA servers host a RentalManager (2nd tier) and RentalDataManager (3rd tier) functionality. The RentalManager handles rental processing logic and customer database updates, while the RentalDataManager handles all data updates associated with rental processing. The motivation for this change is that the architect wants to discover if this splitting of server load will improve performance based on the expected client-request mix. The architect has specified different server-side services to invoke for each of the client

requests, along with a different number of requests for each and differing delays between them. Some clients have a large number of concurrent threads (simulating a larger pool of likely users) while others a small number. The architecture elements, their types and some sample properties for this alternative architecture are shown in Figure 10.

Meta-Type	Element	Sample Attributes&Values (marked with '*')
Client	Client_A* Client_B Client_C	ClientType: CORBA Threads: 15
RemoteRequest	RentVideo * checkCustomer findVideo	RemoteServer :RentalBusinessServer RemoteObject: RentalManager RemoteMethod: rentVideo_service RecordTime: yes TimesToCall: 10
DBRequest	checkCustomer_query findCustomer_query selectVideo_query updateVideo_query * insertRental_query	QueryType :update Dbase: video Table: rental_table TimesToCall: 1 RowsReturned: 1 Caching: no RecordTime: yes
AppServer	RentalBusinessServer * RentalDataServer	RemoteObjs: RentalManager Type:CORBA
RemoteObj	RentalManager CustomerDataManager VideoDataManager * RentalDataManager	Type: CORBA
RemoteService	rentVideo_service* findVideo_service checkCustomer_service findCustomer_service selectVideo_service updateVideo_service insertTental_service	Arguments: 1000 Threading: false ConcurrencyControl: true RecordingTime: yes

Figure 10. Elements, types and example properties for complex video architecture.

7. Generating Performance Test-bed Code

From high-level architecture designs SoftArch/MTE can generate test beds that will provide realistic performance measures for a developed system based on the architecture designs specified. Note that the degree of accuracy will depend on the amount of information the architect specifies and the eventual accuracy of this information. The more precisely the architect identifies likely client->server requests and database updates and the closer the mix of specified requests on servers and databases are the more accurate will the performance test-bed results obtained. We have discovered that architects will usually over-specify requests and database access/update number/frequency to get performance results at the "lower end" of the performance range of an eventual system i.e. will try and get pessimistic rather than optimistic performance results. During architecture design and system development the architect can always update their architecture design in SoftArch/MTE, re-run performance tests and see how the likely system performance may change. Our experiments to date with SoftArch/MTE performance test-bed results and comparison to implemented system performance have shown useful performance estimates are usually achieved from even a very small time investment in high-level architecture design.

The test-bed code generated by SoftArch/MTE is currently "deterministic" i.e. the specified number and ordering of client/server requests, number of threads, pauses between requests, and deployment of clients and servers onto hosts is fixed. The advantage of this is that architects can change parameters associated with architectural elements and the effects

of these changes on performance results are reflected by the changes made to the generated test-bed code. In addition, the more "accurate" an architect wants the test-bed results to be, the more detailed a mix of client and server requests they can specify. The additional test-bed code that is generated will have finer-grained client/server/database code, resulting in both more fine-grained architecture element communication and more detailed performance analysis results for the architect. The disadvantage is that the architect must specify all parameter values for number of users/threads, number of transactions to try, time limit for results capture and so on. They may, of course, get these wildly inaccurate, and will the generated test-bed will capture performance results for this hand-crafted set of interactions. It would, however, be possible with our approach to generate test bed code that created random (within architect-specified ranges) number of threads/users, pauses between requests, number of client requests and so on.

The code generation process used by SoftArch/MTE is outlined in Figure 11. SoftArch/MTE converts the architecture design built by its user into an XML encoding of this design (1). A collection of XSLT transformation scripts and an XSLT engine (2) is used to generate the performance test-bed client and server program source code, IDLs, deployment descriptors, compilation scripts, deployment scripts, database table construction 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).

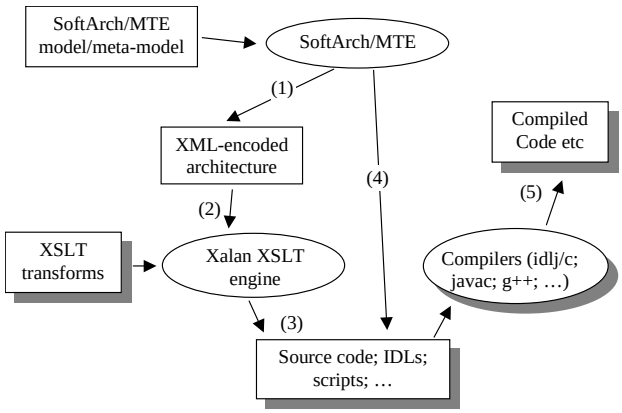
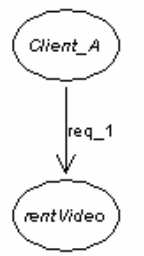


Figure 11. Code generation process.

We chose to use an XML encoding of our SoftArch architecture designs as this provides a design model that can be readily exchanged between tools and be processed by XML-oriented tools. It was natural thus to use a set of XSLT transformation scripts to implement our code generation. The key advantages of adopting XSLT-based code generation from XML-encoded architecture designs include:

- 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
- de-coupling of our code generation and architecture modelling and design representation
- allows new meta-model abstractions to be added using the SoftArch meta-modelling tool that will add new XML items to our architecture representation without breaking the existing code generation;
- allows other CASE tools to import the XML-encoded architecture designs at the beginning full system design/code generation
- will possibly allow other architecture design tools to generate the XML architecture encoding but use our code generation XSLT scripts.

Figure 12 shows part of the architecture model (client request to server remote service) plus examples of XML-encoded architecture design information generated by SoftArch/MTE representing this information. (b) shows the client specification and (c) shows a server remote object specification. The XML represents the architecture elements, their relationships and structural and code generation properties. We found XSLT scripts applied to these XML architecture models provided us with a powerful code generation facility that was significantly easier and quicker to use than using standard programming languages to generate code (as we used in previous CASE tool projects [14]).



Client_A.xml

```
<? xml version="1.0" encoding="UTF-8"?>
   <Client>
     <Name>Client_A</Name>
     <Threads>15</Threads>
     <RemoteRequest>
       <Name>rentVideo</Name>
       <Type>Corba</Type>
       <RemoteServer>RentalBusinessServer</Remote Server>
       <RemoteObject>RentalManager</RemoteObject>
       <RemoteMethod>rentVideo_service</RemoteMethod>
       < TimesToCall >10</ TimesToCall>
       <RecordTime>yes</RecordTime>
     </RemoteRequest>
   </Client>
…
```

VideoServer.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
  <AppServer>
  <Name>VideoServer</Name>
  <Type>CorbaServer</Type>
  <Object>
    <Name>VideoManager</Name>
    <Type>RemoteCorbaObject</Type>
    <ServerName>VideoServer</ServerName>
    <Instances>1</Instances>
    <Service>
      <Name>rentVideo_service</Name>
      <Type>CorbaService</Type>
      <Request>
          <Name>rentVideo</Name>
          <Type>DatabaseUpdate</Type>
          <Status>beginend</Status>
          <TimesToCall>1</TimesToCall>
          <Database>video.mdb</Database>
          <RecordTime>yes</RecordTime>
          <TableName>rental</TableName>
          <RowsReturned>0</RowsReturned>
      </Request>
    </Service>
    <Service>
…
```

Figure 12. (a) Part of architecture model; (b) XML encoding of some of model.

To give an idea of the way we used XSLT scripts to generate performance test-bed code, Figure 13 shows part of an XSLT transformation script used to convert parts of the XML matching CORBA client requests into CORBA client code in Java.

```
<?xml version="1.0"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="xml" indent="yes"/>
<!-- <xt:document href="{Name}.java"> -->
 <!-- generate class header & constructor -->
<xsl:template match="Client">
<MyJava>
public class <xsl:value-of select="Name"/>
{
  public static void main(String[] args){
    int threadNumber=<xsl:value-of select="Threads"/>;
    int index=0;
    while(index!=threadNumber){
      (new <xsl:value-of select="Name"/>_Thread(args)).start();
      index++;
    }
  }
}

class <xsl:value-of select="Name"/>_Thread extends Thread
{
      <xsl:for-each select=" RemoteRequest /RemoteObject">
  private <xsl:value-of select="."/><xsl:text> </xsl:text>
    <xsl:value-of select="."/>_SINGLETON;
    </xsl:for-each>

  public <xsl:value-of select="Name"/>_ Thread(String[] args)
  {
  <xsl:for-each select="/Client/RemoteRequest">
    if((<xsl:value-of select="RemoteObject"/>_SINGLETON) == null)
    <xsl:apply-templates select="RemoteObject" mode="connect"/>
  </xsl:for-each>
  }

  public void run()
  {
    <xsl:value-of select="RemoteRequest/Name"/>();
  }

  <xsl:apply-templates select="RemoteRequest"/>
}
</MyJava>
</xsl:template>
```

```
<xsl:template match="RemoteObject" mode="connect">
    try{
      ORB orb = org.omg.CORBA.ORB.init(args, null);
      org.omg.CORBA.Object objRef =
          orb.resolve_initial_references("NameService");
      NamingContext ncRef = NamingContextHelper.narrow(objRef);
      NameComponent nc =
          new NameComponent("<xsl:value-of select="."/>", " ");
      NameComponent path[] = {nc};
      <xsl:value-of select="."/>_SINGLETON =
          <xsl:value-of select="../RemoteServer"/>Lib.
          <xsl:value-of select="."/> Helper.narrow(
                    ncRef.resolve(path));
    }
    catch(Exception e)
    {
      System.out.println("ERROR : " + e);
      e.printStackTrace(System.out);
    }
</xsl:template>

<xsl:template match="RemoteRequest">
   public  void <xsl:value-of select="Name"/>()
   {
       int iter = <xsl:value-of select="TimesToCall"/>;
       String recordTime = "<xsl:value-of select="RecordTime"/>";
       System.gc();
       long start = System.currentTimeMillis();
       int i=0;
       while(i != iter)
       {
         <xsl:value-of select=" RemoteObject"/>_SINGLETON.
           <xsl:value-of select="./RemoteMethod"/>();
         i++;
       }
       if(recordTime.equals("yes"))
       {
         long time = System.currentTimeMillis() - start;
         double elapse = (double)(time) /
           (double)(Math.max(1,iter));
         String perf = "<xsl:value-of select=".//Name" />
                    _V_V\t"+time+"\t"+iter+"\t"+elapse;
         System.out.println(perf);
         //System.err.println(perf);
       }
   }
</xsl:template>
</xsl:stylesheet>
```

Figure 13. XSLT threaded client code generation script example.

The XSLT script contains Java code fragments that are "fixed" for all clients and XSLT instructions that copy values out of the XML architecture design encoding and generate Java function names, argument values, control construct expression values and so on. Each of the <xsl:value-of select="…"/> constructs is used to extract model element values to parameterise the code generation being done by the script. When each of the XSLT scripts is run for a given input file it substitutes the specified values in the XML file, referred to in the XSLT script by XPath expressions, for the actual architecture design values. This results in Java code files, batch scripts, deployment descriptors and so on being generated specific to the architecture design values given in the XML design encoding.

Figure 14 shows an example of a Java code file implementing a client application program for the performance test-bed for the part of the architecture design and XML client data encoding shown. The relationship between architecture elements in the XML encoding and the resultant client Java program fragment is illustrated. Various parameters specified as properties of the architecture design elements in the XML encoding have been translated into Java code fragment values, statements and classes by the XSLT-based code generation script above. This client can be compiled and run and will perform using multi-threaded clients the specified number of server requests, collecting and outputting the total time taken for these requests for further analysis.

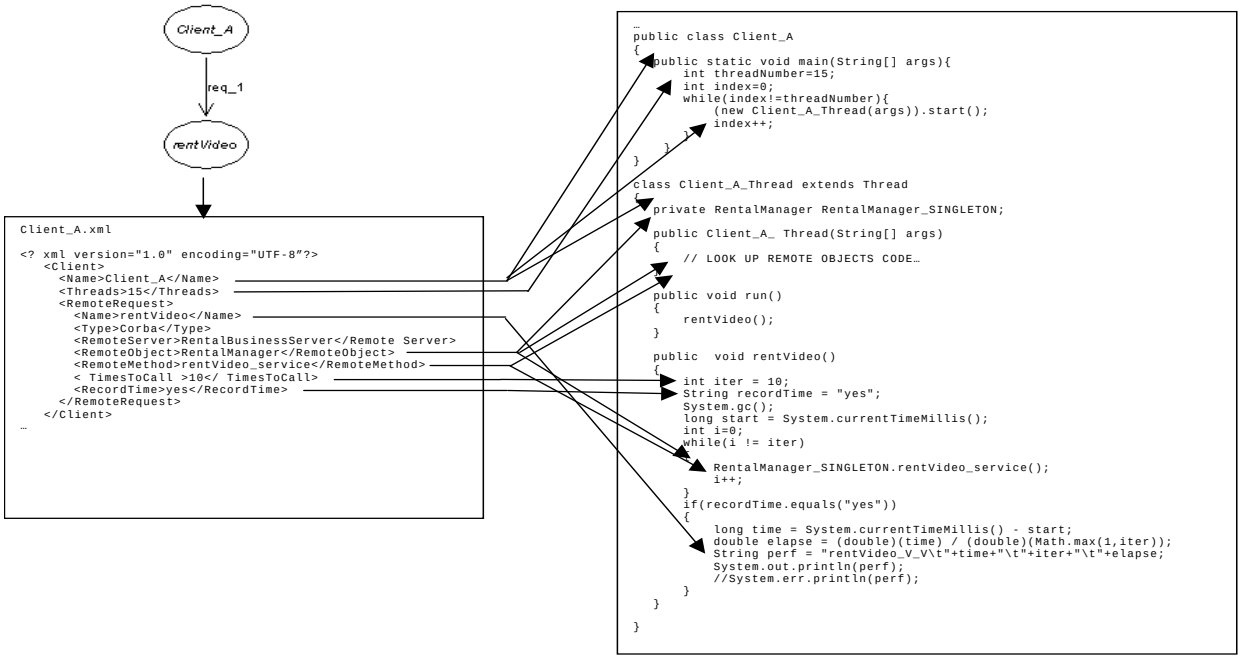


Figure 14. (a) Model XML and (b) example generated Java client code.

Different XSLT transformation scripts can match the same XML-encoded architecture data, generating different code. For example, 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 its XML encoding modification. We have found that this makes extending the code generation facilities of SoftArch/MTE much easier than language-implemented code generation. At present we have code generation scripts that produce CORBA, RMI, J2EE (Enterprise JavaBeans), Java Server Pages, Active Server Pages, ADO.NET and SOAP web service middleware performance test-bed code. All are implemented by different XSLT transformation scripts that are invoked by SoftArch/MTE on generated XML encodings of architecture designs.

8. Testing and Visualising Architecture Performance

Once performance test-bed code and script files have been generated the test-beds must be deployed, tests run and results captured and visualised. Our approach with SoftArch/MTE has been to try and fully-automate this testing and results capture process. A set of available test hosts are identified to SoftArch/MTE by architects – these provide the

machines on which the tests will be run. Our aim has been to focus on providing a realistic deployment scenario where generated client and server and database test-beds can be deployed realistically and thus test results obtained be as accurate as the architecture description and generated test-bed code permits. We developed a "deployment agent" which runs on all available test-bed program hosts and is used to upload information about the host to SoftArch/MTE and to download SoftArch/MTE-generated test bed code and scripts, deploy the test-bed code using the scripts, co-ordinate the running of tests on the host and upload results back to SoftArch/MTE.

Figure 15 outlines our test-bed code performance testing process. Generated code is compiled by SoftArch, using generated compilation scripts (1). The compiled code/IDLs/deployment descriptors etc and the scripts to configure and deploy these on a host are up-loaded to remote client and server hosts using the remote SoftArch/MTE deployment agents (2). The deployment agents organise the uploaded code and associated scripts into suitable directory structures, run the scripts to properly configure the host machine, its database and registries, and to deploy the test-bed code.

The client and server programs are then run: CORBA, RMI and other server programs are started; EJB, JSP and ASP components are deployed into J2EE and IIS servers; database servers started and database table initialisation scripts run; and finally clients are 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. Clients send servers requests, logging performance timing for different requests to a file (4). Servers like-wise log the time taken to execute their remote methods and database operations. Performance results are currently collected in comma-separated value text files. These results are sent back to SoftArch/MTE after tests have completed (5).

SoftArch/MTE collects the test results and aggregates them using simple data processing algorithms to form a unified result set for the performance tests. The results are associated with SoftArch/MTE architecture model instances using data annotation facilities built into the SoftArch modelling tool's repository. SoftArch/MTE then uses data visualisation techniques to display the results of the performance tests to architects, possibly using 3rd party tools like MS Excel™ (6).

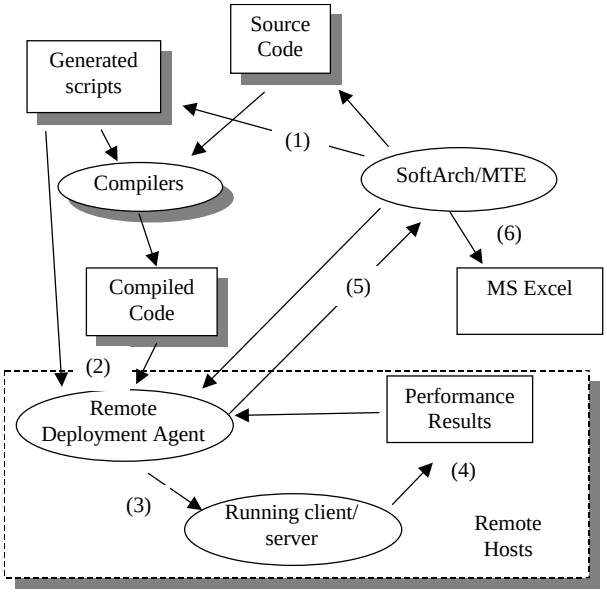


Figure 15. System deployment and test run process.

For thick-client applications the structure of the client-server relationships is straightforward, as illustrated in Figure 16. The client program is started (1) and it looks up the server object(s) it requires (2) which depending on the architecture design may be located on different remote hosts. When instructed by its deployment agent (3) the client runs its specified server remote request operations (4), collects the results of these (5) and the deployment agent uploads the results to SoftArch/MTE (6).

For thin-client applications, such as JSP and ASP-implemented server pages, SoftArch/MTE generates server-side implementations of these using (currently) JSPs or ASPs and these are hosted by a J2EE server or IIS server respectively (1). A "pseudo-browser" client running on client hosts is configured by scripts generated by SoftArch/MTE and run by our deployment tool (2). When instructed to perform its tests the pseudo-browser instructs its IE6 object (3) to request

page(s) from the servers (4), which may in term make database or remote object requests. The pseudo-browser client generates performance results to a file (5) and this data is uploaded to SoftArch/MTE by the deployment tool (6).

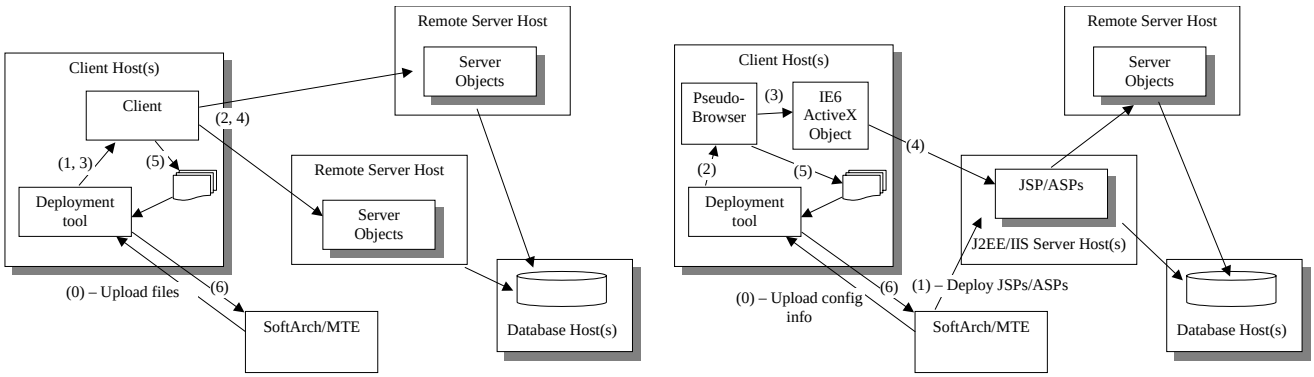


Figure 16. (a) Thick-client test-bed structure; and (b) thin-client test-bed structure.

The performance test-bed results currently include name of request/service, name of owning object/process/program, number of times called, overall time taken, and data stored/retrieved. When collecting this data SoftArch/MTE records performance results against appropriate architecture client requests, services, client/server objects and program elements. Data is summarised across all test bed client and server instantiations to produce an aggregate test result. 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.

Architects need these performance results visualised so that they can determine the overall performance of an architecture, view (where possible) the performance of parts of the architecture e.g. time taken in one server method vs. another vs. in database; and ultimately can compare performance results for different middleware and database selections and ultimately for different architecture designs. In previous work on the MTE project we used graphs to describe hand-crafted performance test-bed results [10]. On the SoftArch project we provided a visualisation mechanism that allowed developers to visualise performance metrics in high-level SoftArch architecture diagrams by the use of annotations [12]. We wanted to reuse these two approaches in order to provide architects with high-level views of performance results obtained from running generated test-beds.

We reused the visualisation techniques described in [12], which were originally developed for real, fully-implemented architecture visualisation, to automatically visualise our generated test-bed results. The architect requests particular performance results they wish to view and a range of architecture design element annotations are used to visually indicate light vs. heavy requesting/loading, small vs. large time consumption, small vs. large data transferral, etc. An example of such a performance visualisation is shown in Figure 17 (1). This example shows an annotated form of the video system diagram from Figure 7 after a performance test has been run. In this example, the annotations are showing total time taken for different architectural elements and their interconnecting relationship. Note that the performance results obtained from client and server objects have been summarised by SoftArch/MTE to obtain aggregate data which is what the visualisation is using to construct its annotations. We wrote a “visualisation agent” for SoftArch/MTE that is plugged into SoftArch and converts the performance results collected by from our generated performance test-beds into the same form generated by our code annotations as described in [12]. This converted data is then used by our existing visualisation algorithm to generate the annotations illustrated in Figure 17 (1).

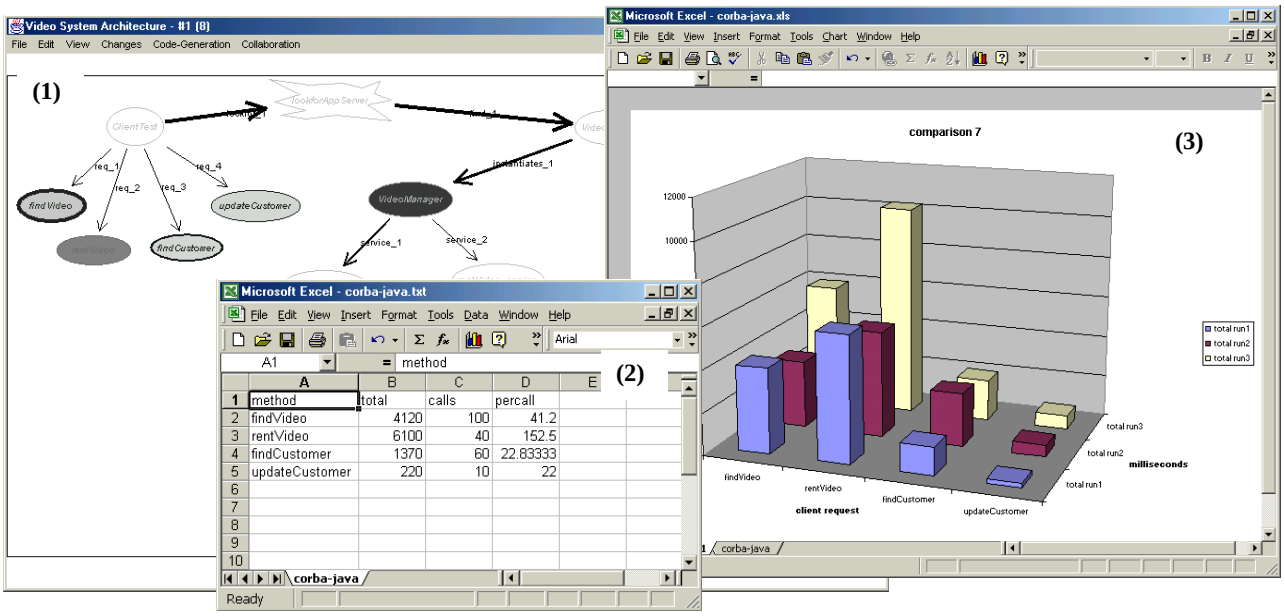


Figure 17. Visualising test run results in SoftArch/MTE and MS Excel™.

SoftArch/MTE performance results are summarised into comma-separated value files by deployment agents. SoftArch/MTE further aggregates results from all deployment agents hosting generated client and server test-bed programs. Figure 17 (2) shows an example of performance result values for one client collected by SoftArch/MTE and opened for display using a MS Excel™ spreadsheet. When results are aggregated into a single result set across all clients and servers collecting performance figures these can be visualised in SoftArch/MTE or in other tools, such as MS Excel graphs. Figure 17 (3) shows aggregated results from three separate performance tests of the video system. Each of these uses a different architecture design for this example system - multiple CORBA objects in one server process; remote objects split across 2 different server processes; and remote objects split across 3 server processes. SoftArch/MTE aggregated the results of generated architecture performance tests into one Excel file and then an MS Excel™ chart was used to chart these values for comparison by the architect.

9. Discussion

8.1. Experiences with SoftArch/MTE

During our research work we have used SoftArch/MTE to help us evaluate the performance of several simple architectural designs for the video system. These designs involved different organisations of server-side functions e.g. single remote object with all functions; multiple remote objects each with different functions; multiple objects deployed on different hosts; and multiple instances of the same object on different hosts. They also involved centralisation of all database information and splitting of database information across multiple database servers and database server hosts; specification of caching for some data; threading and synchronisation choices for remote objects and their services (methods), and choice of different middleware (EJBs, CORBA and RMI objects) and database servers (SQL Server, Cloudscape and Interbase). We also generated thick-client applications (Java applications using remote CORBA, RMI and/or Enterprise JavaBean objects) and thin-client applications (using Java Server Pages and Active Server Pages). The thin-client applications have JSPs or ASPs hosted by web servers (Apache and IIS respectively) and have JavaBeans and .NET components generated to encapsulate data and remote object/database connectivity code.

The SoftArch meta-model and the architecture descriptions for these different client, middleware and database specifications are very similar with subtypes and type properties used by the architect to select different choices. Architecture designs are different assemblies by the architect of client, server, remote object, database, request, service and database table abstractions. For each kind of client, remote server object and middleware/database configuration we wrote XSLT scripts that traverse the XML-encoded SoftArch/MTE architecture specification. These scripts can be easily modified to generate differently-structured code. For example, we changed our thin-client generation scripts from generating stand-alone JSPs and ASPs to generating JSPs that use related JavaBean classes and ASPs that use .NET web

components. This was to produce web server page implementations using more realistic design and implementation structures for testing purposes. The overhead of changing the XSLT scripts was very low with no changes needed to SoftArch/MTE or its XML model to generate these much more complex, fine-grained web server page test-bed implementations.

The main technical challenges we faced when building SoftArch/MTE include identifying and defining suitable architecture abstractions and properties; using XML encodings of architectures to generate suitable test-bed code; capturing appropriate performance measurements from the generated test-bed code; and providing useful visualisations of these measures to architects. SoftArch/MTE allows architects to model a range of high-level and lower-level architectural designs. Our experience with the tool suggests that when an architect can't model the architectural abstractions they want, it is usually due to an incompleteness of the meta-model and its associated code generation scripts. Currently it is not difficult for architects to extend these, but it can be very time-consuming and error-prone. More work is needed both to improve the modelling language of SoftArch/MTE to make it easier to model architectural abstractions and to improve its customisability. Currently only simple generated code instrumentation is used to collect performance results and simple aggregation and visualisation of these results is supported.

8.2. Performance Results

To try and get an idea of the potential accuracy of our SoftArch/MTE performance test-bed generator results, we compared the performance of the video system 3-tier architecture test bed completely generated by SoftArch/MTE to the performance results obtained when running the same client performance tests on a fully-implemented version of the system's servers built as part of a previous teaching project. For this test we used a thick-client (Java application), CORBA middleware and SQL Server 8 database. Several services were modelled, including simple data access (e.g. findCustomer) and simple data updates (updateVideo) that in the real system are passed object parameters and perform simple business logic constraint checking and single database manipulations. Complex data querying (findVideos) and services with complex business logic processing and data update (rentVideo and returnVideo) were also modelled, that in the real system perform multiple database queries and updates. Two sets of results are summarised in Figure 18. One shows averaged results across 5 test runs with 100 requests of each type to the server, using a single-threaded client. The second shows averaged results across 5 test runs for 10 of the same requests when 10 concurrently running clients are used (where other server threads run during network and database blocking). In each case, the same generated test-bed clients were run on both the generated test-bed servers and the hand-implemented servers (with some small hand-modifications to the generated clients to provide appropriate required parameter values for the real servers).

Single-threaded client tests				
	Generated Server (ms)	Actual System Server (ms)	Difference (ms)	% diff
rentVideo	684.4	909.6	225.2	24.75
returnVideo	459.4	640.4	181	28.26
findVideos	227.8	312.6	84.8	27.12
findCustomer	181.6	262.6	81	30.84
updateVideo	252.8	256.4	3.6	1.40
Multi-threaded client tests (10 concurrent clients)				
	Generated Server (ms)	Actual System Server (ms)	Difference (ms)	% diff
rentVideo	435.28	631.42	196.14	31.06
returnVideo	433.48	603.6	170.12	28.18
findVideos	157.18	220.64	63.46	28.76
findCustomer	154.26	202.08	47.82	23.66
updateVideo	202.86	225.84	22.98	10.17

Figure 18. Results of generated test-bed server code performance vs. implemented server performance.

Client request	Actual system performance result	Generated System performance result	Difference (ms)
productManagePage	27ms	26ms	-1
accountManagePage	23ms	24ms	+1
orderManagePage	39ms	28ms	-11

Figure 19. Results of C#.NET-implemented Pet Shop application tests.

We also modelled the architecture of the Pet Shop reference architecture and compared a C#.NET version (a thin-client architecture) to a freely-available 3rd party implementation of this system developed by Microsoft Corp. using C# and ASP.NET. In this system, the servers include web page content construction (via ASPs) as well as database access, update and simple business logic. The clients were run with 8 concurrent threads and no pause between requests to the servers. Results are shown in Figure 19. Again we compared the results obtained running the same SoftArch/MTE client performance testing code on both generated test-bed server code and existing system hand-implemented server-side code. For the real Pet Shop system we again had to add by hand some example URL argument data to the ASP server page calls from the clients to ensure they worked correctly.

In the above tests, the performance of the generated test-beds to the hand-implemented systems was generally quite close. When a small number of server-to-server or server-to-database requests are made by the server-side abstractions, the results are generally very close, as in the Pet Shop example. With larger number of complex server-side services, such as the video system's rent/return video processing, the results can vary due to business logic in the real system which is not present in the generated test-beds, and due to additional synchronisation code. For example, the generated orderManagePage service in the Pet Shop example above lacks some synchronisation code that is in the real system, resulting in the test-bed performance being artificially very high. In addition, in both examples the test bed does not pass complex datasets between requesting objects and service. The performance impact of this is noticeable in the findVideos and findCustomer examples for the video system, where these services have multiple arguments in the real implementation.

We have also used SoftArch/MTE to help us model some other system architectures and compare the performance of different architecture and middleware design choices. We designed architectures for a micro-payment system [6] that we have implemented as part of another research project which uses CORBA, J2EE JSPs and EJBs and relational databases. We modelled an on-line travel planning system used on previous research and teaching projects: one using a peer-to-peer decentralised RMI architecture, one using Java Server Pages and JDBC, one using Active Server Pages and ADO.NET, and a final variant implemented using C# and .NET SOAP web services. We have also designed simple SoftArch/MTE architectures for an enterprise workflow system and an integrated health informatics system (which provides a range of heterogeneous clients and servers using EJBs, XML and CORBA). With each of these systems we modelled a number of remote server interfaces using SoftArch/MTE and a mix of thick- and thin-client user interfaces. In most cases we modelled simplified forms of the architectures of the fully-implemented systems, with fewer client requests and server services than in the target systems.

In general, the anecdotal performance results obtained when running the fully-generated test-beds for most simple client requests and data-oriented server processing functions have been of a similar degree of closeness as the above examples, to those obtained by running the same client test programs against the manually-implemented servers. However, complex client requests and server services with program logic we can't currently model with SoftArch/MTE e.g. complex algorithms and decision logic affecting the number of subsequent remote object calls and database updates. These can thus provide very inaccurate results. In addition, SoftArch/MTE provides abstractions describing client/server/database request interactions an architect can only use SoftArch/MTE to model an estimated number of remote requests and data updates/accesses for such services. These may well be very inaccurate and be very different in the final system implementation. However, SoftArch/MTE can still be used to obtain performance information incrementally during development with the architect refining the estimated number of flow-on requests for a service to produce more useful performance estimates.

8.3. Evaluation of Our Approach

Key advantages of the SoftArch./MTE approach to performance test-bed generation include the ease with which architects can specify an architecture design and have realistic performance results obtained; the extensibility of our code generation and meta-modelling approach to supporting this performance test-bed generation; and the use of real code and

machines to run the test-beds, thus obtaining "realistic" results. Our approach allows results like those obtained from hand-coded architecture and middleware performance analysis projects [10, 17, 27] to be obtained with much less effort on the part of developers. In addition, a major advantage is that developers can change their architecture designs and middleware choices, having their performance test-beds completely automatically regenerated and performance tests re-run. We have incrementally enhanced our toolset by adding new meta-model abstractions and code generation support (e.g. for modelling and generating thin-client applications using JSPs and ASPs) without impacting on existing architecture models and SoftArch architecture tool implementation. XSLT has proved to be a very flexible, powerful tool for implementing our extensible code generation facility from XML-encoded architecture models.

Our current approach to performance test-bed generation has some key limitations. The architect is constrained to use the provided meta-model abstractions and XSLT code generation scripts. In order to use other middleware, they or others must extend the SoftArch/MTE meta-model and its code generation scripts, the latter essentially programmatic effort. This work can be challenging e.g. it took us several weeks to build working EJB code generation and deployment scripts and around three person-months to build basic JSP and ASP thin-client generation scripts. As it is often hard for an architect to estimate the likely mix of client requests, database access and update requests and server-to-server requests in a final system when doing architecture design, such inaccurate loading specifications will naturally produce inaccurate performance measures. We had this experience when specifying architectures for our travel planning system and forgetting to account for some common client requests and server processing scenarios. This resulted in very unrealistic performance results being obtained compared to running the same client tests on parts of the fully-implemented system.

However, it must be emphasised that **all** testing approaches (prototypes, simulation and even monitoring fully developed systems) suffer from this same problem when using estimated loading rather than using real users and real data. Thus any of these approaches to estimating system performance based on a certain architecture design is only that - an estimate. We have noticed the ease in specifying discrete client and server requests in SoftArch/MTE architecture designs does usually lead to architects, especially novices who have tried using the system, specifying overly-simple designs. However, we suggest tools like SoftArch/MTE not be used as a once-off architecture performance test-bed generator but have an evolving architecture design kept up-to-date with development work on a project. The architect needs to continually refine their model and more fine-grained client-server requests be specified as the architecture design evolves, re-running performance tests periodically to track changing likely eventual-system performance hot-spots.

An additional requirement of our approach is the need for a reasonable number of client and server host machines to be available to run test-beds on in order to obtain meaningful results. 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 due to the need to co-locate some client and/or server test-bed code that in reality would not be run on the same machine or use the same kind of inter-machine network. In such scenarios, an architecture performance simulation technique may be able to produce more accurate performance estimations by modelling large numbers of heterogeneous client and server hosts that are not available for a SoftArch/MTE user.

8.4. Future Research

Currently SoftArch/MTE generates a performance test-bed using fixed values for architecture structural and code generation properties. The performance results obtained are fixed values of number of transactions performed or time taken for the transactions to complete. A possible extension would allow architects to specify ranges of values e.g. 9-12 concurrent threads; repeat request 4-6 times; and so on. SoftArch/MTE could run then run the tests with different values for parameters and report the results as value ranges. Alternatively it could select random values within the ranges and report results, possibly after running the tests several times. These approaches would provide software architects with ranges of performance results, including maximums, minimums and standard deviations. These may be more useful for architecture design performance analysis than the current fixed-value results reported at present.

We have been adding new code generation scripts to SoftArch/MTE and meta-model abstractions to support generating message-oriented middleware (e.g. Java Messaging Service and MQ Series) code and are looking to generate code for SOAP (web services) middleware, both for Java and .NET platforms. We are have implemented some basic C# code generation scripts in order to performance test architecture designs with this intended target implementation language. We are working on improved, richer performance result visualisations, including showing results for different test runs on (slightly) modified, overlapping architectures together. Currently SoftArch/MTE uses its own architectural representation in XML. We are investigating using a "standard" XML-based architecture design encoding, like xADL [21] or the XMI UML encoding, allowing other tools to use our code generation and deployment scripts.

We have recently been extending the open-source UML design CASE tool Argo/UML [31] to provide a SoftArch-style modelling diagram and meta-modeller for these diagrams, with extensions to the XMI encoding used by Argo/UML

to represent these architecture designs. We have extended the UML's support for architecture modelling to enable the specification of SoftArch/MTE-style architecture designs with test-bed code generation properties. This tool provides users with a SoftArch/MTE-style performance test-bed generator within a more common CASE environment. We plan to test this environment with industry co-operation on some large enterprise system performance analysis and tuning projects. We are also planning to investigate the extraction and update of architecture designs within this extended Argo/UML environment from Argo's OOD designs. This will allow users to automatically update their architecture fine-grained client requests and server services from an evolving system design, running re-generated test beds to determine how a system with an evolving, partially-implemented architecture is likely to perform if implementation is completed.

10. Summary

We have developed a prototype performance test-bed generator that automates the generation of performance prototypes, running of prototypes and visualisation of test-bed prototype performance using high-level software architecture models. Our SoftArch/MTE tool provides a high-level, extensible architectural modelling language that is rich enough to allow fully-working test bed code to be generated. We encode the Softarch/MTE architecture designs in XML and use a set of extensible XSLT transformations scripts to transform an XML-encoded architecture design into test bed client and server program code and compilation/deployment scripts. A deployment agent running on available client and server hosts is used to automatically upload compiled systems and to configure and deploy them. Test runs are performed and the performance results captured by client and server programs are automatically captured and aggregated by SoftArch/MTE. These results are visualised by either the annotation of in the high-level architecture design diagrams within SoftArch/MTE or by 3rd party applications like MS Excel™.

We have used Softarch/MTE to model a number of distributed systems, generate performance test-beds for these models, capture results and compare these results to performance measures obtained from hand-implemented, completed distributed systems. Our experiences have demonstrated that our approach provides a useful, accurate automated architecture performance analysis environment for these distributed systems. Main limitations of our approach include the need for a number of client and server host machines in order to get a realistic distribution of client and server test-bed programs, and the need for architects to identify and specify the "key" client requests and database processing for the architecture in order for useful results to be obtained. Use of SoftArch/MTE within a development process where architecture designs are continually updated and test-beds re-generated and tests re-run allows architects to incrementally refine their architecture designs and remain aware of the likely performance of a completed system based on this design. A tool like SoftArch/MTE can thus be used as a key enabler of proactive architecture design and performance-directed architecture evolutionary development.

Acknowledgements

Parts of this work were supported by the University of Auckland Research Committee, the New Zealand Public Good Science Fund and the New Zealand New Economy Research Fund.

References

1. Balsamo, S., Simeoni, M., Bernado, M. Combining Stochastic Process Algebras and Queueing Networks for Software Architecture analysis, In *Proceedings of the 3rd International Workshop on Software and Performance*, Rome, Italy, July 214-26 2002, ACM Press.
2. Bass, L., Clements, P. and Kazman, R. *Software Architecture in Practice*, Addison-Wesley, 1998.
3. Beaumont, M. and Jackson, D. Visualising Complex Control Flow. In *1998 IEEE Symposium on Visual Languages*, Halifax, Canada, September 1998, IEEE.
4. Chen, M., Tang, M. and Wang, W. Software Architecture Analysis - A Case Study, In *Proceedings of COMPSAC'99.*
5. CSIRO, Middleware Technology Evaluation project, www.cmis.csiro.au/mte.
6. Dai, X. and Grundy, J.C. Architecture for a Component-based, Plug-in Micro-payment System, In Proceedings of 5th Asia-Pacific Web Conference, September, Xi'an, China, Springer LNCS.
7. ECPerf Performance Benchmarks, August 2002, ecperf.theserverside.com/ecperf.

8. Egyed, A. and Kruchten, P., Rose/Architect: a tool to visualize architecture, In *Hawaii International Conference on System Sciences*, Jan. 1999, IEEE.
9. 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.
10. Gorton, I. And Liu, A. Evaluating Enterprise Java Bean Technology, In *Proceedings of Software - Methods and Tools*, Wollongong, Australia, Nov 6-9 2000, IEEE.
11. 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.
12. Grundy, J.C. and Hosking, J.G. High-level Static and Dynamic Visualisation of Software Architectures, In *2000 IEEE Symposium on Visual Languages*, IEEE.
13. 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.
14. Grundy, J.C., Mugridge, W.B. and Hosking, J.G. Constructing component-based software engineering environments: issues and experiences, *Information and Software Technology: Special Issue on Constructing Software Engineering Tools*, Vol. 42, No. 2, January 2000, pp. 117-128.
15. Grundy, J.C., Cai, Y. and Liu, A. Generation of Distributed System Test-beds from High-level Software Architecture Descriptions, In Proceedings of the 2001 IEEE International Conference on Automated Software Engineering, San Diego, CA, Nov 26-29 2001.
16. Hill, T., Noble, J. Visualizing Implicit Structure in Java Object Graphs, In *Proceedings of SoftVis'99*, Sydney, Australia, Dec 5-6 1999.
17. 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, pp. 270 – 276.
18. Juiz, C., Puigjaner, R. Performance modelling of pools in soft real-time design architectures, *Simulation Practice & Theory*, vol.9, no.3-5, 15 April 2002, Elsevier, pp.215-40.
19. Jurie, M.R., Rozman, I., Nash, S. Java 2 distributed object middleware performance analysis and optimization, *SIGPLAN Notices* 35(8), Aug. 2000, ACM, pp.31-40.
20. Kazman, R. Tool support for architecture analysis and design, In Proceedings of the Second International Workshop on Software Architectures, ACM Press, 94-97.
21. 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.
22. Lee, S.C., Offutt, J. Generating test cases for XML-based Web component interactions using mutation analysis, In Proceedings of the 12th International Symposium on Software Reliability Engineering, Hong Kong, China, 27-30 Nov 2001, IEEE CS Press.
23. 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.
24. Liu, A. Dynamic Distributed Software Architecture Design with PARSE-DAT, In *Proceedings of Software – Methods and Tools*, Wollongong, Australia, Nov. 2000, IEEE.
25. 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.
26. Ma, Y-S. Oh, S-U. Bae, D-H., Kwon, K-R. Framework for third party testing of component software. In Proceedings of the Eighth Asia-Pacific Software Engineering Conference, IEEE CS Press, 2001, pp.431-434
27. 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, 1998, pp.180-185.
28. Microsoft Corp., Using .NET to implement Sun Microsystem's Java Pet Store J2EE BluePrint application, October 2002, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/psimp.asp.

29. Nimmagadda, S., Liyanaarachchi, C., Gopinath, A., Niehaus, D. and Kaushal, A. Performance patterns: automated scenario based ORB performance evaluation, In Proceedings of the Fifth USENIX Conference on Object-Oriented Technologies and Systems, USENIX, 1999, pp.15-28.
30. 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.
31. 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.
32. Shannon, B., *Java 2 platform, enterprise edition : platform and component specifications*, Addison-Wesley, 2000.
33. Shaw, M. and Garlan, D. *Software Architecture*, Prentice Hall, 1996.
34. 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.
35. Vogal, A. CORBA and Enterprise Java Beans-based Electronic Commerce, *International Workshop on Component-based Electronic Commerce*, Fisher Center for Management & IT, UC Berkeley, 25th July, 1998.
36. Woodside, C. Software Resource Architecture and Performance Evaluation of Software Architectures, In *Proceedings of the 34th Hawaii International Conference on System Sciences*, IEEE, Maui, HA, Jan 2001.