System Factory: Integrating Tools for System Integration

Vadim Kotov
Computer Systems and Technology Laboratory
HP Laboratories Palo Alto
HPL-1999-118
October, 1999

E-mail: kotov@hpl.hp.com

system integration, system modeling, system analysis, customization, simulation, analytic modeling, Petri Nets, Communicating Structures, System Factory

Large highly distributed systems such as multi-tier server farms, global enterprise systems, E-commerce and E-service systems are both critical for businesses and complex and expensive to build and run. Their design requires an advanced technology of system integration that guarantees satisfaction of various customer requirements such as cost/performance, high availability, recoverability, security, etc. This report advocates a shift from uncoordinated tools, which are developed independently to address specific tasks, to integrated environments that incorporate various system analysis techniques and tools and may be customized for the design of such IT systems that fully satisfies the whole spectrum of customer requirements.

The functionality and structure of a System Factory prototype is described in the report. The center of the factory is a Repository that accumulates system solutions, patterns, and templates for specific business segments. Specialized Shops with a common modeling and analysis base provide convenient means to find customized solutions. Communicating Structures, a uniform and systematic representation of large systems as hierarchical, distributed, and communicating subsystems and components, provides this common base.

Internal Accession Date Only

CONTENTS

1. INTRODUCTION

Rapidly converging computer and communication technologies and, specifically, explosive growth of net-enabled technologies bring us to a new IT landscape. It is mainly formed by complex systems that integrate hardware, software, networks, middleware, applications and data into multi-tiered systems, ranging from clusters of servers up to enterprise-wide computing infrastructures and global massively distributed internet-based systems. These systems, or rather *Systems of Systems*, consist of large number of components of a vast variety: from high-end servers and huge databases to laptops, handholds, personal appliances, and mobile devices. They have sophisticated connectivity and high traffic densities, implement newest technologies of storing, manipulating, and delivering of various types of data, including complex texts, voice, graphics, and video. As a result, the variety of feasible system architectures and their complexity are rapidly increasing. The main problem of these systems is that though even they are built of highly reliable and efficient components they may exhibit bad global behavior and performance due to bad application and data partitioning, wrong traffic organization, inappropriate security policies, etc.

Systems of Systems represent a challenge for the design and integration. On one hand, the solution space is huge and complex. On the other hand, these systems are increasingly critical for businesses. The cost of their deployment and management amounts to a quite substantial part of corporate budgets. It is very difficult to pick good solutions that are adequate to growing customers' requirements.

To meet the new level of system complexity and to deliver much better tailored systems and E-services than the competition, system and service providers need an *advanced system integration technology* that should be focused at *mass customization* as an important market differentiator. Mass customization in system integration will produce and deliver systems that are designed to fit specifications of individual customers, are state-of-art of IT, and are, simultaneously, cheaper than "manually" integrated systems.

Such a technology should be a breakthrough in system design, integration, reengineering, and management. It should not only radically speed up system delivery, increase cost/performance, reliability and security of delivered systems, but also reduce cost of ownership and improve system manageability and reactivity to business.

System Factory is a metaphor that being implemented provides such a technology in the form of an efficient working environment for system designers and integrators. It includes various modeling and analysis tools for addressing various aspects of system design, a repository for accumulating system patterns, templates, and solutions as well as specialized shops supporting different stages of the system analysis, modeling, synthesis, and management.

Why a factory is better than just a tool set? The factory

- moves the system integration from craftsmanship to industrial setting,
- emphasizes priority of systematic quantitative methods of analysis and taking decisions,
- promotes standards and well-defined interfaces for data exchange between different tools,
- supports the usage of patterns, templates, and prefabricated solutions and, as a result, speeds-up (time-to-market!) the system integration process and makes it more reliable,
- provides a common frame to incorporate quickly new tools,
- builds-up a common logistics for user interface, visualization, and documenting.

The evaluation of solutions is done at the factory using multiple criteria and requirements (performance, reliability, high-availability, …) and optimization may be made in one (but multi-dimensional) metrics.

The center of the factory is *Repository* that accumulates system solutions, patterns, and templates for specific business segments. Specialized *Shops* with a common modeling and analysis base provide convenient means to find customized solutions. *Communicating Structures*, a uniform and systematic representation of large systems as hierarchical, distributed, and communicating subsystems and components, currently provide this common base.

2. CONSOLIDATION AND MASS CUSTOMIZATION

The spectrum of IT-enabled businesses and variety of corporate IT infrastructures are growing. Customers are interested in system consolidation to avoid complexity and cost of proprietary system deployment and management. System integrators are also interested in consolidation, as integration from scratch has many disadvantages: long time-to-market, significant cost of design, and not well-tested *Quality of Solutions*.

To maximize the return from investment into IT, customers expect that systems, while satisfying traditional cost/performance and Quality of Service requirements (low response time, high-availability, security…), will additionally be:

- "ready-made" integrated systems customized and tuned for their business processes,

- scalable and modifiable enough to respond to changes in businesses, and

- reactive to special (critical) workload situations.

As special orders and made-to-order systems cost more, the customers look for systems that tailored to their needs – at the lowest possible price. In other words, they look for customized systems.

Business processes in different industries have both common and specific features. The whole space of business processes may be partitioned into *business segments* of related business models. In the same way, the space of IT infrastructures that support different business models may be ramified into correspondent *IT models, or patterns*. Finally, implementation of the IT patterns results into a branching structure of typical *system solutions* (see Figure 1). Such a ramification may have several layers, each layer being a refinement of the previous one.

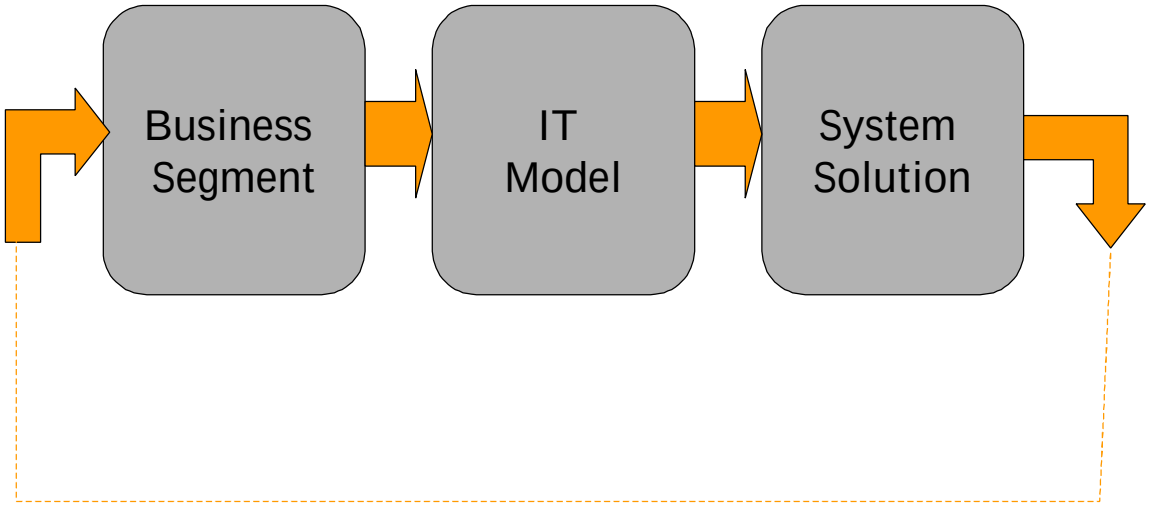

Figure 1.

After we have structured businesses and IT patterns, we can, for each business segment and correspondent IT model:

- identify a typical *generic system configuration*, or a *template system*,
- identify the main parameters that influence cost/performance, availability and other QoS,
- identify the segment's workload patterns,
- analyze the solution space for the template system,

- find the best (generic) solution(s) for the segment,
- accumulate these solutions for the further usage,
- identify typical *sub-segments* satisfying specific system QoS requirements and accumulate correspondent template systems and solutions for the further usage,
- continue the process to satisfactory granularity of segments.

Having a hierarchy of solutions accumulated in a solution repository, a system integrator can start his design with identifying an IT segment in the repository that may contain solutions close to the desired ones. The next phase is the customization of the template architecture. It requires a new analysis of the solution space under specific constrains or modifications of IT processes and requirements, but this time the space is much smaller and chance to make a serious design mistake is smaller.

3. SYSTEM FACTORY FUNCTIONALITY

System Factory is a working environment for system designers and integrators that should help them quickly deliver quality systems. System integrators can tailor their "virtual systems" using customer requirements and the System Factory as a smart configurator. They can experiment with "virtual systems", analyze them, and elaborate the best solutions, which then will be implemented in the real world systems.

In its advanced form, *System Factory* supports three phases in the system life cycle, as shown in Figure 2:

- design and integration;
- updating and reengineering;
- management and maintenance.

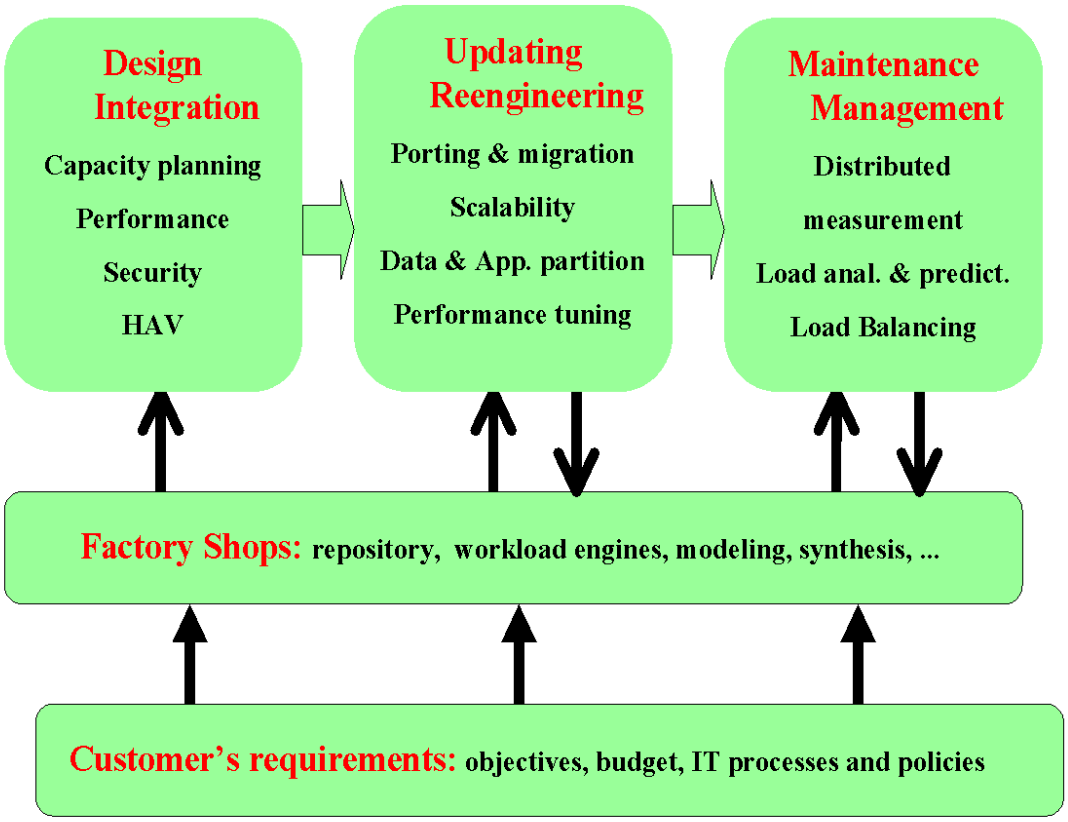

Figure 2.

The *Design and Integration* phase is the first and the main phase at which the future system emerges. The phase consists, in its turn, of several stages:

Qualifying and Evaluation

- an information on goals and objectives of the design is obtained, QoS requirements, the system scale and budget are studied,
- business and IT processes to be supported by the designed system are analyzed,
- applications characterization and profiling of a potential workload are made,
- components characterization and users special requirements are analyzed.

Modeling and Analysis

- an appropriate system segment is chosen,
- the architecture of the template system of the chosen segment is modified, refined, and scaled,
- the system components (hardware, software, …, applications) are selected,
- models of the selected components and the designed system model(s) are built or retrieved from the *Factory Repository* (see below),
- the designed system model(s) is (are) constructed form the component models and template system model(s) stored in *Factory Repository*,
- if it is possible, workload measurements are made and analyzed; workload is synthesized, otherwise,
- the model(s) is (are) used to make capacity planning, to analyze performance under various projected workloads, to identify potential bottlenecks,
- the model(s) is (are) used to study scalability, availability and other requirements,
- measurements and experiments are made to validate and calibrate the models,
- search for best solutions is made using the validated model(s),
- proof of the concept, benchmarking, testing and verifying complete the stage.

At the "*Updating and Reengineering*" phase, the models built at the previous stage are used to find the best way to upgrade the designed system if the IT requirements or capabilities have been changed due to substantial changes in business processes or/and in technology. *System Factory* may be used to check how the system QoS may change as a result of

- significant increase in the number of system subsystems and components,
- significant increase of the permanent workload,
- addition of new types of system components,
- porting and migration of new applications or databases,
- changes in data and application partitioning among the system's subsystems,
- planned phased conversion from an old system to some newer system.

Then *System Factory* uses the obtained information to find what changes in the original design should be made to tune the system performance and other Quality of Solutions parameters.

The "*Management and Maintenance*" phase uses the model(s) built at the *Design and Integration* stage to manage dynamic system features, such as load analysis, prediction, and balancing. Distributed agents can be used to trace changes in the system (new nodes, retired nodes, some topology and parameters changes, new applications, etc.) and to monitor workload. The model parameters are automatically updated when the agent's report changes. The updated model is analyzed, and the results are forwarded to the system manager (or managers) for taking some appropriate actions to redistribute workload, add resources, etc.

4. SYSTEM FACTORY STRUCTURE

The *System Factory* consists of the following *shops*:

- Repository,
- Modeling Tool Kit,
- Specialized Shops.

4.1. Repository

Repository is an intelligent database (or a product management system for systems) that stores information obtained and used in the system design and integration, namely information about

- basic system components (hardware, software, networks, middleware, applications) used in the design,
- typical template models,
- typical solutions,
- typical workload patterns, traces,
- benchmarks.

The whole information in Repository is structured into (a hierarchy of) segments, related to different business patterns and correspondent IT infrastructures. The segments may also differ by levels of detail. XML [Har99] is an appropriate specification language for *Repository* as it provides standard and contemporary technology to work with complex structured data. It is an unambiguous, extensible, platform independent language.

4.2. Modeling Tool Kit

Modeling Tool Kit is collection of tools and libraries for

- simulation of large and complex systems,
- analytical modeling (capacity planning, simulation validation, …) ,
- high-availability and recoverability tools,
- security analysis tools.

To make System Factory not just mere a collection of tools, but an interactive synergetic federation of problem solvers, it is necessary to provide (1) a common system specification basis for different shops and tools, and (2) standardized interfaces for interaction between its shops, tools, and repository.

Communicating Structures Library (CSL) developed at HP Labs for the high-level representation, modeling and analysis of large communicating systems form a basis for *Modeling Tool Kit*. It provides both simulation and analytical capabilities and is able to emulate other modeling paradigms, such as Petri Nets. The whole diversity of system components and structures is represented in CSL as a uniform and systematic composition of a small number of simple basic concepts that describe data traffic and data placement in systems.

4.3. Specialized Shops

These shops specialize on different aspects of system design and integration. Figure 3 exemplifies the basic set of shops.

Workload Engine provides capability to analyze traces, received from *Repository* or from *Measureware* Shop, or synthesize artificial workload and feed it into *Model Runner*.

Model Builder provides the possibility to conveniently describe system topologies, system parameters, as well as variations in topologies and parameters in both textual and GUI form.

Measureware collects resource and performance data from components and subsystems (applications, databases, middleware, networks, and OS. The collected data are stored either in *Repository* or fed directly into models.

Model Runner actually prepares and runs modeling experiments specific for system integrator's objectives and to used tools.

Solution Analyzer analyzes modeling results and presents them in forms convenient either for human perception or for *Solution Synthesizer*. Visualization of a solution space is extremely

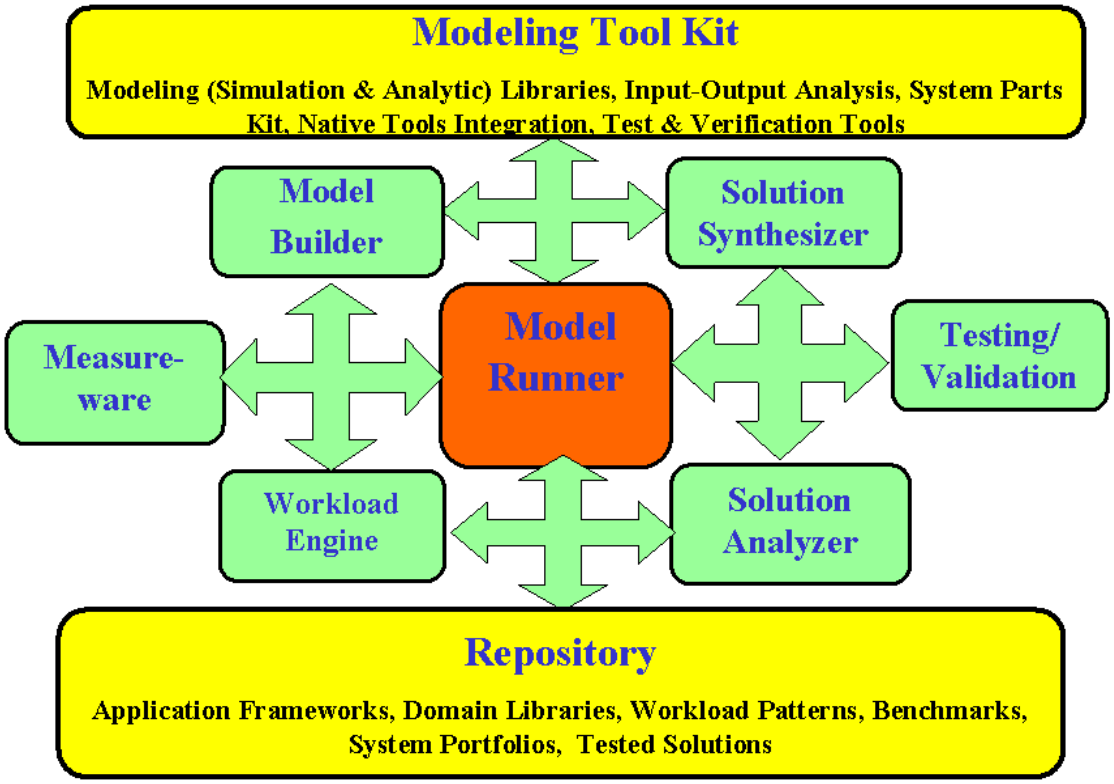

Figure 3.

important for the analysis of the modeling results. Due to the necessity to provide multi-variant optimization of solutions, *n*-dimensional visualization may be needed.

Solution Synthesizer looks for better solutions in the Sea of Solutions typically available after modeling of complex systems with multiple optimization criteria.

Testing/Validation Shop uses benchmarks and results of experiments and measurements with template systems stored in *Repository* to test and validate the delivered customized systems. Alternative configurations may be finally delivered for further considerations.

4.4. Factory Framework

All *Factory* shops are put together in the framework of an object-oriented environment that allows user to visit separate components, work within them, exchange data among shops and between *Factory* and external world. The environment is implemented in Java, is platform

independent and provides GUI for navigation in *Factory*. Figure 4 shows *Factory* when the *Model Builder* shop is visited. The left panel contains the textual (XML) specification of a (E-service) model, which is constructed by a user from scratch or is customized (using a template model from *Repository*). The right panel displays a fragment of the model communication structure for visual control and graphical editing.

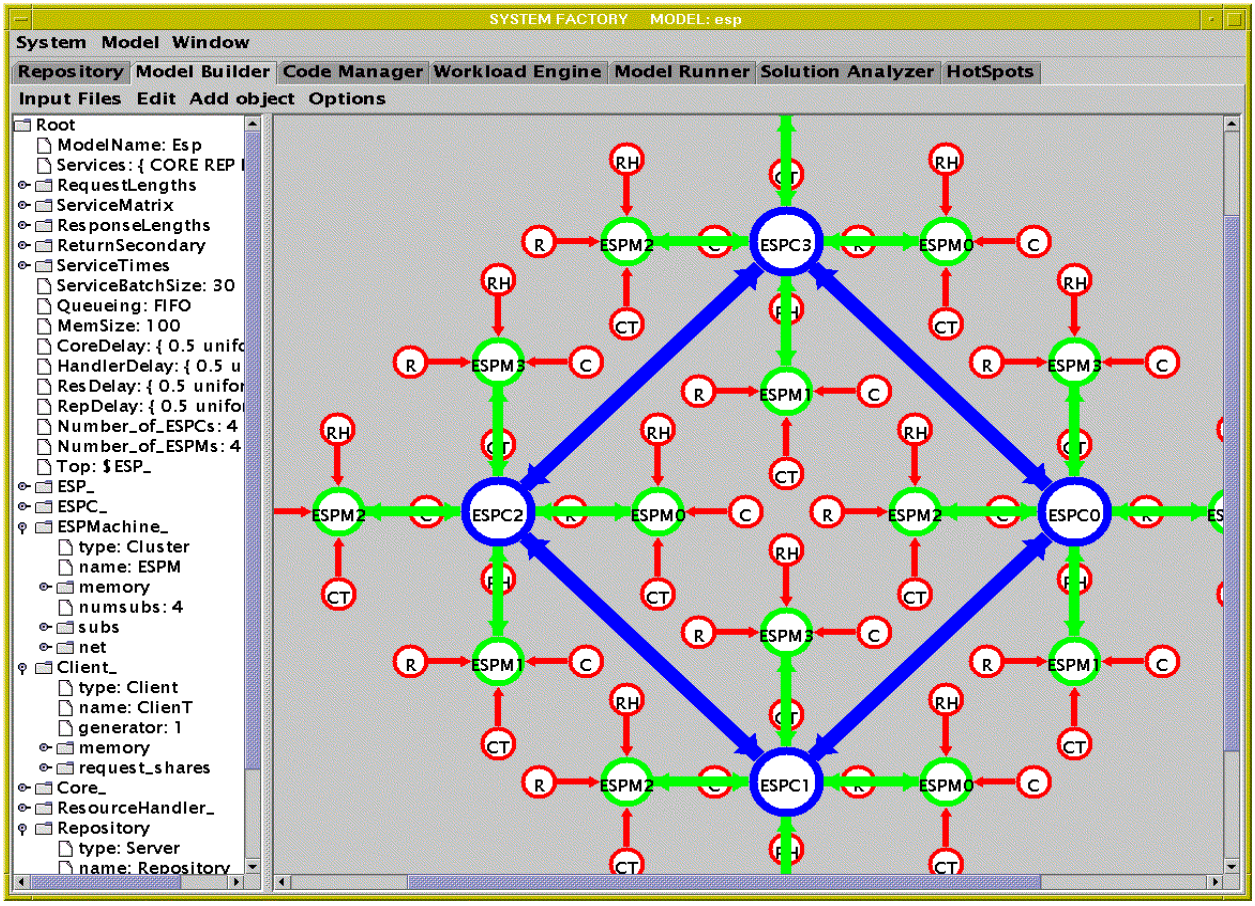

Figure 4.

.

The Figure 5 shows an example of the visual analysis of “hot spots” provided by the *Solution Analyzer*.

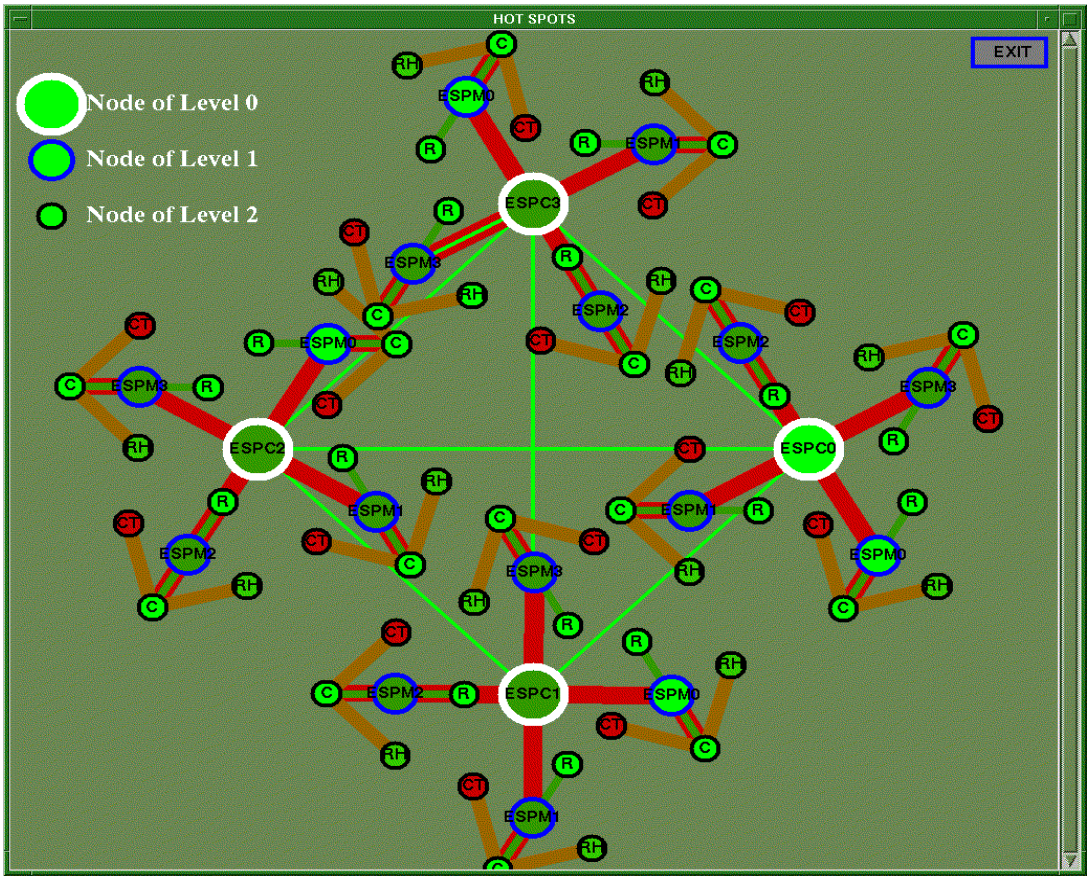

Figure 5

5. COMMUNICATING STRUCTURES LIBRARY (CSL)

Communicating Structure [Kot98] is a hierarchical distributed structure that represents the system components and the communications between them in a uniform and systematic way. The system components are represented simply as *nodes*. Each node has *memory* that may contain *items*. *Nets* are sets of links that connect the nodes. The items are generated at some nodes and move from node to node along links, with some delay. Nodes may modify items. The item traffic models the data traffic in a system, which is represented as a communicating structure. (Communicating Structures were proposed to model systems with traffic-sensitive performance in the first place.)

Items, nodes, memories, and nets may be elementary or may be aggregate and have some structure. For example, an item may represent simple data such as a word, a frame, a packet, as well as a complex message with large chunks of data or even a transaction consisting of many messages. The nodes may represent "atomic" units or larger aggregate system components. Nets may represent simple point-to-point links as well as busses, crossbars, interconnects, cascaded switches, LANs, communication lines and WANs.

Communicating Structures are high-level, template-style representations of systems in the same sense as template vectors represent vectors of anything. They do not fix behavioral or implementation semantics of basic objects, attributes, functions and process, but rather provide default semantics for them. That makes the possible to "emulate" other conceptual models such as queuing networks or Petri nets. Thus, Communicating Structures may serve as a common base for the implementation of higher-level tools as shown in Figure 6.

Communicating Structure Library (CSL) [KotRok Cher98] is an object-oriented implementation of Communicating Structures. The CSL objects may be assigned different attributes (numbers, variables, functions, and processes) that serve to refine these objects, customize them, supply them with external data, change their behavior and navigate among different levels of modeling detail.

5.1. Simulation

In the case of simulation, the CSL uses the main structures of C++/CSIM, a process-oriented discrete-event simulation package [Sch95].

CSL nodes can be assigned *processes* that are invoked to generate items, receive/send items from/to other nodes, and to transform the items, if necessary. Such a node becomes a *simulation node* and is able to model active components of systems.

Any CSL object is also a *facility*, a resource with *servers*, which can be *reserved*, used (seized), and *released* by processes.

Processes exchange information with other process via *mailboxes*. Processes can send messages to a mailbox and can receive messages from mailboxes. A mailbox has a queue of messages waiting to be received and a queue of processes waiting to receive the messages.

Synchronization and control of interactions between processes is supported by the mechanism of *events*. A process that encounters a *wait* statement with a given event as an argument either continues, if the event is in the *occurred* state, or queues on the event if that is in the *not occurred* state.

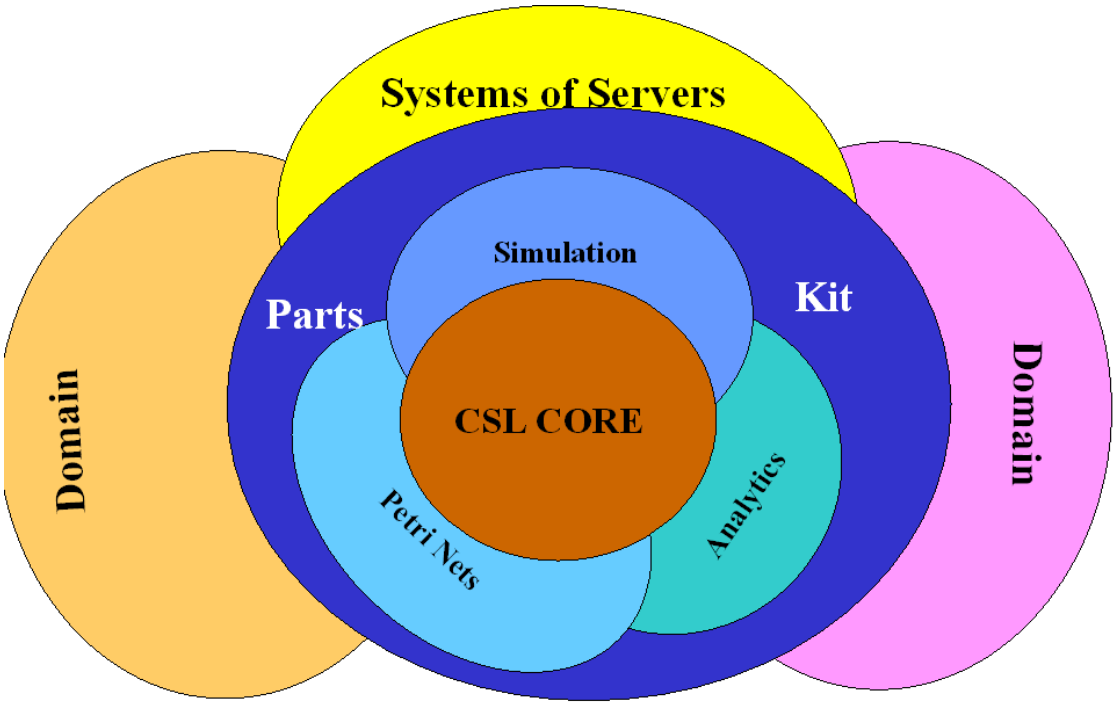

Figure 6.

5.2. Analytic Modeling

CSL allow us to construct also queuing networks [Tan95]. A CSL node and a CSL net (more often, a link) can be presented as a ``service center" or as a ``queue node". Such a queue node executes a queuing model that is associated with it. The type of the model is defined by the arrival time distribution and by the service time distribution, plus by the number of servers in the node. The input data for the queuing model are an interarrival rate and a mean service time. The model returns the average waiting time (different approximations), the average time spent in a queue node, the average number of items in the node, the average number of waiting items, and their standard deviations.

Given

- the number of the queue nodes and,
- for each queue node,

- the arrival rate from outside the network,
- the probability that an item goes from this node to another given node,
- the service time,
- the number of servers,

the CSL-based queuing net returns for each node:

- the average time spent in a queue node,
- the average number of items in the node,
- the average number of waiting items, and
- their standard deviations.

5.3. Petri Nets

The CSL nodes represent both places and transitions of Petri nets. In case of a transition, it is an “active” simulation node. In case of a place, it is a "passive" node with no processes associated with it. The current number of servers at the facility of a node-place shows the current number of tokens in this place.

Petri net arcs are represented by the CSL links. The multiplicity of an arc is transformed into the number of the servers at the correspondent link.

To implement the firing conditions at transitions, each node-transition waits to reserve its input nodes-places with the number of requested servers at each node-place equal to the multiplicity of the link-arc connecting this transition with this place. When the transition fires, it decreases the number of servers at its input nodes-places and increases the number of servers at its output nodes-places.

In a similar way, CSL can simulate *Colored Petri Nets*. To express token colors, *Colored Facility* is introduced as an associative array of facilities with colors serving as array's keys. In other words, the facilities and their servers are assigned colors. The regular facility *reserve* and *release* operations are extended to discriminate the colors of servers.

6. DOMAIN LIBRARIES

CSL inherits all object-oriented mechanisms of C++. They allow us to introduce user-defined classes of nodes, memories, nets, items, and other constructs and to accumulate them in a

hierarchy of libraries with increasing specialization. Examples of such libraries are *Systems of Servers* and *Multi-Tiered Systems.* Examples of such libraries are described in this section.

6.1. Systems of Servers

One of such "domain" libraries called *Systems of Servers* has been constructed on the top of CSL to model distributed server systems and was used to model enterprise-wide IT infrastructures of large corporations. Main objects of *Systems of Servers* are:

- services,
- servers,
- clusters of servers (and of clusters of servers),
- clients (proxies),
- messages,
- transactions (or sessions).

Servers, clusters and proxies are refined CSL nodes; messages and transactions are refined CSL items.

Examples of problems that can be addressed in *Systems of Servers* are:

- comparison of topologies of distributed servers,
- partitioning of services among servers in clusters of servers,
- cashing strategies,
- queuing and scheduling policies,
- load balancing,
- admission control.

The *Systems of Servers* library provides a choice of algorithms for partitioning, load balancing and caching that might be appropriated for different traffic patterns.

6.2. Multi-Tiered Systems

The *Systems of Servers* library serves as a platform for the next level library called *Multi-Tiered Systems* and used to model E-commerce and E-service architectures. A typical generic system in the *Multi-Tiered Systems* library consists of up to 4 tiers (Figure 7):

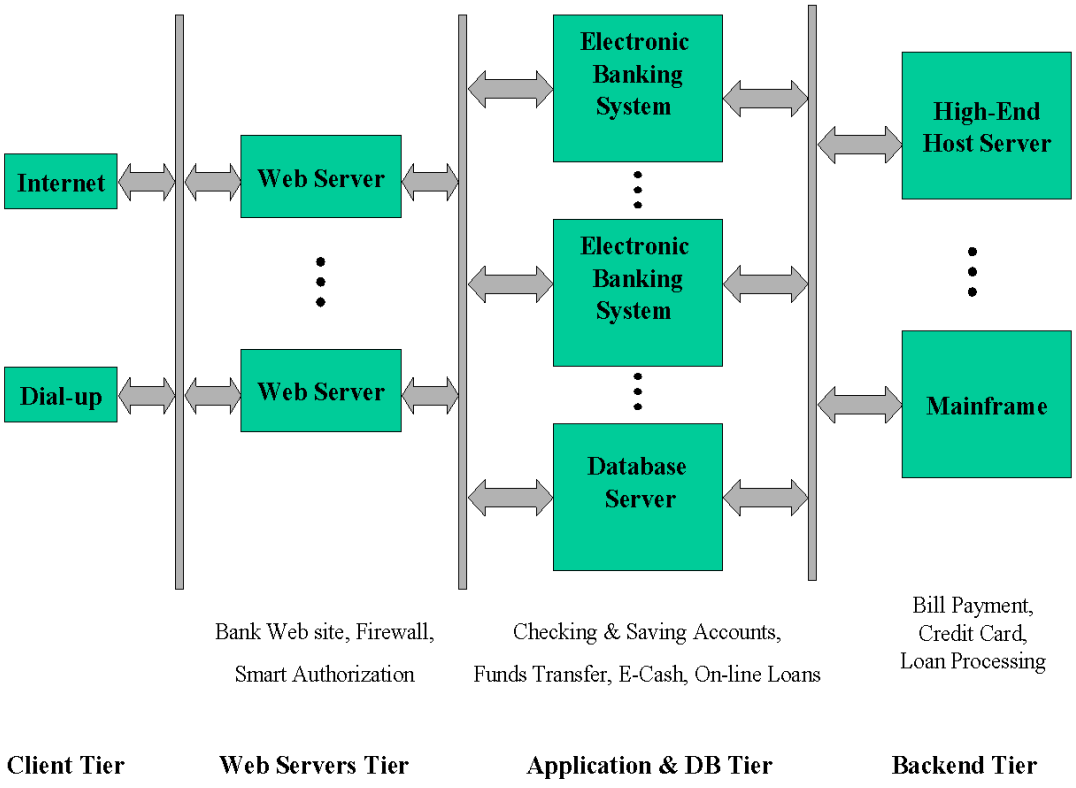

Figure 7.

- Tier 1: *Thin clients.* They are external Internet, ATM, and phone/fax clients accessing the banking system for services via transactions handled by these services.
- Tier 2: *Front-end (Web) services and servers.* It provides a secure foundation for the applications that Internet-enable an enterprise. It connects outside clients to the "inside" servers via an internal secure boundary.
- Tier 3**:** *Application services and servers.* It provides self-service solutions deployed at several application and DB servers. Transactions are packaged into modules that access the services.
- Tier 4: *Back-end services and servers.*

Generic template model, built around this sort of architectures with the help of *System Factory*, may be easily customized for specific electronic delivery services in banking or retail businesses.

Each tier contains relatively small number of services and servers. Thus, the service-server matrix in *Repository* may be not too large. Similarly, the security and availability requirements do not vary significantly for different customers. The main variable parameters are partitioning services among servers (primarily inside tiers) and request traffic patterns and traffic density that may challenge scalability.

6.3. Template Models

Most cases, in which the CSL–based libraries were used for the validation of distributed system architectures, have one thing in common. Before solving a specific system design or integration problem, a generic, template-like model has been built. This model represented a wider class of systems (and potential solutions) than it was required. Then, this generic model was customized for the analysis of a particular case under consideration. That allowed us to extend the range of the system analysis and extend the solution space. The customization was typically carried on "manually". It is assumed that it will be made "more automatically" in *System Factory*. There is no sharp distinction between a library and a template model; this is a matter of emphasis on generality.

6.3.1. Distributed Shared Memory Multiprocessors. For example, a class of distributed shared memory multiprocessors has been modeled using a template model. Each of these systems represented some number of aggregated nodes interconnected by some switch fabric. Each node includes either some number aggregated subnodes or it consists of processors with caches. Some type of bus interconnects the processors each with others and with a shared memory of the node. The problem was to investigate different distributed memory coherence policies for different system configurations, different interconnect topologies and their parameters. The template model represents a hierarchy of processing nodes with some given total number of processors at the bottom. The following parameters can be varied:

- the number of hierarchy levels,
- the distribution of the nodes among the hierarchy levels (e.g., 4 x 4 x 4, or 8 x 4 x 2, or 8 x 2 x 4),
- the various types of interconnects (stars, rings, buses, …) , their throughput and latency at each level of hierarchy,
- different workload patterns.

Some smart strategies of navigating the solution space help to do this faster for larger total number of processors. Such an approach allows to have in *Repository* prefabricated tables or spreadsheets that allow a system integrator to evaluate the bottom line parameters of typical system topologies and then to refine them for particular cases of workload.

6.3.2. Mission-Critical Systems. A project of a worldwide distributed system for a global transportation company has been analyzed using a generic CSL model that may serve as a template model for mission-critical company-wide IT infrastructure for delivery companies.

The distributed system does:

- tracing and monitoring of packages (hundreds of millions of transactions per day),
- statistics, billing data processing, and decision support ,
- customer services (including WWW),
- common enterprise business applications.

The system represents three-level hierarchical network of three-tiered computing centers:

- global *Data Centers*, several of them,
- regional *Processing Centers*, tens of them,
- local *Operations Centers*, tens of thousands of them,
- mission critical, ``almost real-time" computing environment,
- cost effectiveness is the dominant requirement.

So, this is a typical *System of Systems* and it was modeled in CSL in a natural way.

Information is distributed and exchanged among centers according to the *Publish-Subscribe* paradigm: applications publish data for potential use by other applications and are subscribers for data published by others. *Point-to-Point* dispatch and *Data Brokerage* are two alternative models for the implementation of the *Publish-Subscribe* methodology. The first case represents a spider web of point-to-point interfaces, which are "hard-coded" with specific languages, platforms, application and data formats. Applications maintain unique relationships between themselves. In the second case, point-to-point links are replaced by the publication of common messages usually in a standard format, which are sent to *Data Brokers*. The *Brokers* have the tables of subscribers for each type of the messages and forward the messages to subscribers. The task was to evaluate the project with the emphasis on comparison of the two *Publish-Subscribe* models.

The constructed CSL model contains those and only those system features that influence the message traffic and are important for satisfying the global system requirements. The model helped to identify bottlenecks and the system sensitivity to changing parameters (the number of centers, bandwidth in local and global networks, message packaging principles, publish-subscribe mechanisms, etc.).

The sensitivity and utilization analysis included:

- system response on burst workload,
- system scalability,
- *Data Broker* overhead,
- message packaging strategy,
- utilization of *Data Brokers* as a function of workload,
- utilization of different levels of network.

The main result of the project validation was the reduction of the proposed three-level system architecture to two-level architecture. The CSL analysis of the traffic in the system has shown that if the functions of the second level are redistributed between the top level of Data Centers and the low level of the Operation Centers, then the global traffic becomes less congested, response time is improving, basic requirements to the system are satisfied and the overall cost is, of course, dramatically reduced.

Such a model may serve as a prototype template model for mission-critical company-wide IT infrastructure for global delivery companies.

6.3.3. Global Enterprise Systems. The most complex case study was the validation of the enterprise-wide IT infrastructure of a big corporation The main task was to predict the system scalability with the number of workstations changing in the range from several 1000s to several 10,000s.

The system has four tiers: workstations, application servers, method servers, and database servers. (Method servers provide support for the PDM (Product Data Management) programs that access and manage data in database servers.) The traffic of requests goes from the first tier to the last one, and the response traffic goes back. The requests have different priorities and responses have different length, the request-response traffic may be quite burst. Again, to deal with such a complex system, a generic model was constructed that actually has been converted and extended into the *Systems of Servers* library.

7. CONCLUSION

System Factory is not just a toolbox. This is a rigorous and systematic way to built and upgrade systems with emphasis on objective, quantitative selection of best solutions. The *Factory* itself will evolve with time, addressing yet new problems and challenges emerging in large-scale systems that form the backbone of modern information technologies. As any factory, *System Factory* will influence the way in which the system components are designed and build,

improving their quality, modularity, and fitness to integration.

8. ACKNOWLEDGEMENTS

Thanks to Holger Trinks, an intern from the Chemnitz University, who actively participated in the preparing of the second release of the System of Servers Library. In particular, he analyzed and implemented some load balancing and caching policies. He also built the multi-tiered system template model (section 6.2).

Thanks to Igor Tatarinov, a summer intern from the North Dakota University, designed and implemented the first prototype of the Swing GUI for System Factory.

9. REFERENCES

[Har99] Harold, E.R. XML Bible. IDG Books Worldwide, 1999.

[Kot98] Kotov, V.E. Communicating Structures for Modeling Large-Scale Systems. In Proceedings of the 1998 Winter Simulation Conference, Washington, D.C., ed. D.J.Medeiros, E.F.Watson,J.S.Carson, and M.S.Manivannan, 1998, pp.1571-1578.

[KotRokCher98] Kotov, V.E., Rokicki, T.M., and Cherkasova L.A.CSL: Communicating Structures Library for System Modeling and Analysis. HP Labs Technical Report HPL-98-118,Palo Alto, CA, 1998.

[Sch95] Schwetman, H. Object-Oriented Simulation Modeling with C++/CSIM17. In Proceedings of the 1995 Winter Simulation Conference, Washington, D.C., ed. C. Alexopoulos, K. Kang, W. Lilegdon, and D. Goldsman, 1995, pp. 529 – 533.

[Tan95] Tanner, M. Practical Queueing Analysis. McGraw-Hill, 1995.