A Framework for Analyzing and Organizing Complex Systems

Sven Graupner, Vadim Kotov, Holger Trinks
Computer Systems and Technology Laboratory
HP Laboratories Palo Alto
HPL-2001-24
February 6th , 2001*

E-mail: {sven_graupner, vadim_kotov, holger_trinks} @hp.com

distributed systems, system management, system model, optimization, model-based management, system factory

The paper discusses a framework and advanced technologies enabling the quantitative analysis, organization and optimization of large-scale, globally distributed enterprise and e-service systems.

The goal is to organize complex systems in such ways that traffic at application and service layers can be better explained, predicted and controlled. In distinction to traffic management at the network layer, our work approaches higher system perspectives where architectural decisions are made about the overall organization of work and task flows, the global placement of data and applications, caching and replication etc. Those decisions are significant for the traffic induced in the system later on. Little support is provided today for designing and evaluating large-scale systems from these perspectives, primarily caused by the difficulties in developing realistic computerized models reflecting dynamic characteristics of services, applications, access patterns, resource demands and capacities.

Novel approaches presented here have been developed to formalize models providing the basis for analysis and understanding large-scale systems from top-level perspectives. The notion of 'systems of systems' refers to viewing systems from different perspectives and capturing the various aspects at different levels of granularity. Case studies with earlier versions of our approach have been carried out with two big corporate partners.

* Internal Accession Date Only Approved for External Publication

A Framework for Analyzing and Organizing Complex Systems

Sven Graupner, Vadim Kotov, Holger Trinks
Hewlett-Packard Laboratories, 1501 Page Mill Road, Palo Alto, CA 94304, USA
{sven_graupner,vadim_kotov,holger_trinks}@hp.com

Abstract

The paper discusses a framework and advanced technologies enabling the quantitative analysis, organization and optimization of large-scale, globally distributed enterprise and e-services systems.

The goal is to organize complex systems in such ways that traffic at application and service layers can be better explained, predicted and controlled. In distinction to traffic management at the network layer, our work approaches higher system perspectives where architectural decisions are made about the overall organization of work and task flows, the global placement of data and applications, caching and replication etc. Those decisions are significant for the traffic induced in the system later on. Little support is provided today for designing and evaluating large-scale systems from these perspectives, primarily caused by the difficulties in developing realistic computerized models reflecting dynamic characteristics of services, applications, access patterns, resource demands and capacities.

Novel approaches presented here have been developed to formalize models providing the basis for analysis and understanding large-scale systems from top-level perspectives. The notion of 'systems of systems' refers to viewing systems from different perspectives and capturing the various aspects at different levels of granularity. Case studies with earlier versions of our approach have been carried out with two big corporate partners.

1 Introduction

Network management systems can exactly measure and analyze traffic in complex environments at the network layer. Since networks are shared among large numbers of applications, it is hard to associate the observed traffic with individual applications or even higher notions such as services in large-scale distributed systems. Similarly, metrics used in system management basically refer to utilization of individual machine parameters such as CPU and storage. They are also hard to correlate with distributed applications typically sharing machines at geographically distributed locations. It is hard to physically localize distributed applications and associating them with individual machines. Current system management approaches have weaknesses in deriving useful information from their information bases about the behavior of systems at higher system perspectives. Available information is too detailed, hard to correlate and thus too complex for such perspectives.

Another observation from analyzing current system management technology (see also [10]) is:

- component views dominate with a focus on physical (or hardware) components, originating from SNMP-approaches, but also favored by newer extensions such as CIM [1] and XML/CIM [2],
- information models of system management systems are based on compositions of physical components.

As mentioned, it is not favorable attempting to determine physical components participating in performing distributed applications or even higher notions of services comprised by numbers of applications. Referring to physical components just leads to enormous complexity. Dynamism of applications and services actually prevents associating individual portions of them with physically localizable hardware components.

In conclusion, there is a mismatch in what we call granularity and abstraction between complex software systems as targets of consideration and physical component views provided by management systems. Appropriate counterparts for software components need to be identified as "virtual" execution platforms. The layers of software systems (or software stacks) must be complemented with an adequate hierarchy of layers of execution platforms. Higher-ordered layers of execution infrastructures are themselves comprised by distributed software systems. The hierarchy of software layers starts with the local operating system software providing the execution platform for local applications and continues further up the hierarchy of layers. We extend this view and develop a systematic methodology uniformly applicable up to highest system perspectives to large-

scale, globally distributed enterprise and e-services systems. Despite of all differences of components considered at different layers of granularity, we believe that their general behavior follows similar patterns based in queuing theory and traffic flow systems. We make use of this assumption and describe models of individual system layers in similar and compatible terms allowing us to correlate models for various evaluations. It will also allow us moving from component-centric views towards system views with better understanding and predicting system behavior and interaction between systems.

Challenges addressed in this paper are:

- developing a general methodology for identifying and uniformly reflecting components and systems at different layers of granularity;
- identifying the appropriate information to be represented in the model bases of individual layers used for analysis and decision support for overall arrangements in the organization of individual system layers and in the overall system; examples of such decisions are where shares of services or application functionality will be provided, cached or replicated in an effective manner;
- representing (or computerizing) the identified information providing the models for supporting the various stages of systems' life cycles with issues of:
    - how to obtain the model information,
    - how to describe the model information,
    - how to keep the model information current,
    - how to incorporate dynamic parameters,
    - how to maintain statistical information;
- developing tools for monitoring and analyzing systems from higher system perspectives and providing decision support at organizational level – in our approach based on reasoning upon model descriptions representing components at appropriate layers of granularity.

The paper is organized as follows: after briefly discussing the need for viewing and modeling systems from higher system perspectives to support the life cycle of systems throughout all stages, the methodology of viewing systems as *Systems of Systems* is introduced. In this view, systems are considered as being composed of sub-systems representing the various layers of different granularity. Then, the principles for modeling we use are explained.

In the second part of the paper, the *System Factory* is presented as a general framework comprising the various technologies we have developed. Two case studies then show the current stage of investigation and experimentation using that technology.

This report presents the current state of research of the 'Systems of Systems' project at Hewlett-Packard Laboratories, Palo Alto.

2 Supporting the Life Cycle of Systems

As there is a software life cycle, a system life cycle also exists. Our technology and the System Factory framework support the three phases in the system life cycle: design and integration; updating and reengineering; and management and maintenance.

Phase I: Design and Integration

The Design and Integration phase is the first and the main phase at which future system emerge. The phase consists of several stages:

Qualifying and Evaluation:

- the design goals are determined with requirements, system scale and budget considerations,
- business and IT processes are analyzed,
- application characterization and profiling of a potential workload are made,
- component characterization and users' special requirements are analyzed.

Modeling and Analysis:

- an appropriate system segment is chosen,
- the architecture of a template system of the segment is chosen, then modified, refined, and scaled,
- the system components (hardware, software, applications, services) are determined,
- models of the designed components are built or retrieved from System Factory's Repository,
- the designed system models are constructed form the component models and template system models stored in System Factory's Repository,
- workload predictions are made and analyzed; workload is synthesized if not available otherwise,
- models are used for capacity planning, to analyze performance under various projected workloads, to identify bottlenecks prior to implementation.

Model Validation and Calibration with the real System:

- models are used to study scalability, availability and other requirements,
- measurements and experiments are made to validate and calibrate the models,
- search for best solutions using the validated models,
- proof of the concept, benchmarking, testing and verifying complete the Design and Integration stage.

Phase II: Updating and Reengineering

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 changed. System Factory can be used to evaluate how the system may also change as result of:

- significant increase in the number of systems, sub-systems and components,
- significant change of 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.

System Factory uses the obtained information in its model bases to identify what changes in the original design should be made in order to tune the system performance or adjust other parameters.

Phase III: Management and Maintenance

The Management and Maintenance phase uses the models derived at the Design and Integration stage to manage dynamic system features, such as load prediction and balancing. System Factory's distributed monitoring infrastructure can be used to trace changes in the system (new nodes, topology and parameter changes, new applications, etc.) besides monitoring workload and parameters. Model parameters are automatically updated when such changes are reported from the monitoring infrastructure. Updated models are constantly analyzed. Results are propagated to system managers for calibrating model parameters with system observations.

3 Methodology

This section explains what we understand as appropriate layers of granularity for viewing systems as *Systems of Systems*. It also explains the general approach of matching demand occurring at each layer with capacity offered underneath. This principle is uniformly applied across all layers of consideration. It allows us to reduce and consolidate the complex correlations among large-scale application and services' systems into one, characterizing correlation: matching demanded with available capacity in the three dimensions: processing, storage, and transmission. It forms the basis for optimizing general arrangements in the overall system organization at the respective layers.

3.1 Viewing Systems as 'Systems of Systems'

Understanding the behavior of large-scale application systems requires consideration of what we call components of equivalent or matching granularity by vertically classifying systems into:

- *services* at the top layer (=sets of co-operating distributed applications solving tasks),
- *distributed applications* (=sets of application tasks performed at geographically dispersed locations),
- *application tasks* assigned to individual locations representing shares of applications performed there,
- *individual processing* locations of application processes as lowest resolution layer.

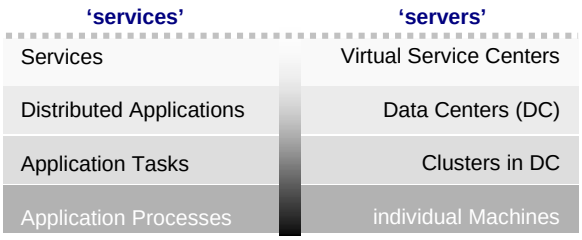

Figure 1: '*Systems of Systems'* view with layers

Accordingly, matching granulates on execution side are:

- *virtual service centers* (as representatives for execution environments assigned to sets of services and located in several, distributed data centers),
- *data centers*,
- *clusters* of machines hosting individual application tasks within data centers, and
- *individual machines* as locations of final processing.

Other classifications may be defined as well to correlate adequate elements in models. The two hierarchies represent counterparts of elements with corresponding granularity for each layer: software systems executed by respective execution environments. We also refer to this as notion of abstracted *'services'* performed by abstracted *'servers'* [3] in order to unify terminology for components at the various layers.

3.2 Resource Demands and Capacities

The primary question in understanding systems is the understanding of resource demands on the one side and available capacities on the other side. Again, components of adequate granularity must be correlated in order to avoid mismatches as pointed out in the introduction. We introduce four layers of granularity for identifying, modeling and correlating components.

Two basic approaches can be applied. One is matching the counterparts of abstracted 'services' with matching 'servers' at each layer (Figure 2). This view reflects the distribution of portions of services, distributed applications, applications tasks and processes among respective processing locations of virtual service centers, data centers, clusters of machines or even refined to individual machines within clusters. At each horizontal layer, the environments of the servicing stations or 'servers' provide the capacities to meet the demand of the 'services' at the respective layer.

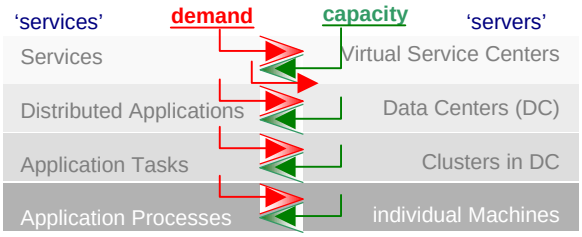

Figure 2: 'Services' distribution among 'servers'

Another approach reflects the vertical resolution of 'services' through its own layer hierarchy and the resolution of 'servers' through the layers on the other side. In this view, demand is induced from the top layer down the hierarchy, and capacity is provided in a bottom up direction – applicable for both sides (Figure 3).

For the 'services' side, the use of services translates into certain activity (or demand) in the underlying layer of distributed applications causing activity in individual application processes which finally translates into load or traffic in processing locations and networks. Capacities satisfy demands in the opposite direction.

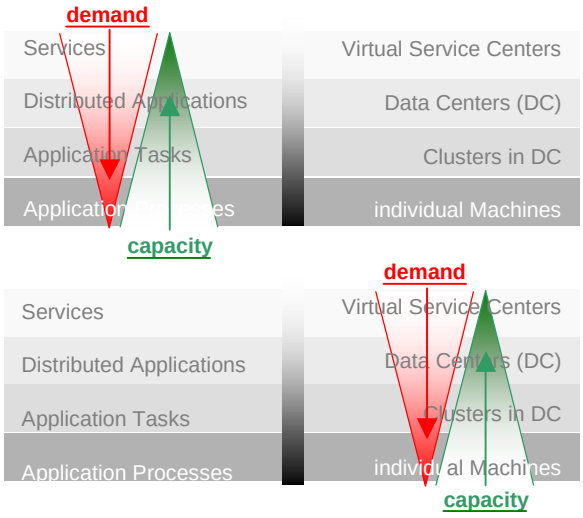

Figure 3: Vertical resolution though layers

Models represent environments for each layer and for each side. They may be correlated with neighbored models in the layer above or below or at the other side in order to formulate various scenarios, views or considerations. Multiple models may be involved in such scenarios, and each model can represent either role. For example, multiple distributed applications are performed in one data center. Their added demand then represents the portion of capacity the data center has to meet. Or, one global service is represented in various data centers around the world. Each of the data centers then absorbs that portion of demand services need in each data center.

One set of models always represents the demand side and counterpart models are representing the capacity side in each correlation. Each model may be referred to in either role and must hence provide both information of the environment it describes in a comparable manner.

In Figure 4, example 1 represents a correlation of mapping services' demands with capacity a virtual service center provides (a virtual service center represents shares of execution environments located in several data centers). The second example shows a correlation within the 'servers' hierarchy of how individual machines provide the capacity of a cluster. Any other neighbored correlation would be possible.

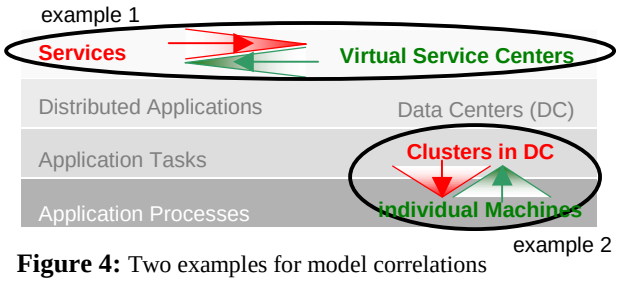

Figure 4: Two examples for model correlations

By manipulating models, various scenarios may be evaluated, optimized and validated by simulation before final decisions are made and implemented in real systems. Capacities may be planned and evaluated as well as demand predictions can be fortified, all based on information about the system represented by models.

Applications or services behave well and can be operated efficiently when the overall system is in a balance throughout all layers. Bottlenecks and congestions may occur at any layer degrading the overall system performance: in the network infrastructure with routers and name servers, at the application layer while accessing a shared data base or at the services layer while performing intermediary services such as a financial transaction. There is no integrated approach today to capture and correlate a complex system vertically through different layers as it is proposed here, and applying similar patterns to description, modeling and analysis for each of the layers (queuing theory, flow analysis etc.).

3.3 Optimal Matching Demand with Capacity

As mentioned, the basic methodology in our approach is the (optimal) matching of resource demand with offered capacity at adequate layers. Models are represented for each side of the introduced layers. For example, there are models describing a distributed application and others describing applications tasks. There may also be models describing the infrastructure installed in a data center in such terms that this can be interpreted as operating infrastructure offering the needed capacity to perform the applications. There is always the duality of resource demand and matching capacity in our approach.

Based on model descriptions, the placement of services, distributed applications, and application tasks can be analyzed and optimized. Results from this optimization provide support for higher-level decisions about the overall system organization. The section 'Generic Optimization Framework' explains how optimizations are performed. Results have been shown in [7].

"Compatible" model representations enabling the correlations and the model bases will be described next.

4 The Information Base: Model Descriptions

To enable the mentioned correlations and optimizations, models have to be "compatible" and composable on the fly by applying the same principles and patterns representing their environments. These principles are explained here. After that, the System Factory framework is discussed we are using for experimentation.

In the following, the requirements for model descriptions are outlined allowing the correlations in the ways described:

- Compatibility of model descriptions including referring to the same elements and relationships used for representations in model descriptions and for parameter sets. A generic language is needed. Traditionally, a Lisp-like language is used (see fragment in Figure 6) which can be translated into an XML representation we use externally.
- Common parameter sets have to be identified expressing respective demands and capacities for each model in a generic way.
- Constraints need to be expressed in a generic manner.
- Incompleteness of information must be tolerated. We do not assume the presence of model descriptions for all layers and environments, only for those of interest.

The last point is in particular important since our targets are large-scale systems whose dynamism prevents completeness of information at each time. Our models thus represent rather statistical behavior reflecting the essential characteristics.

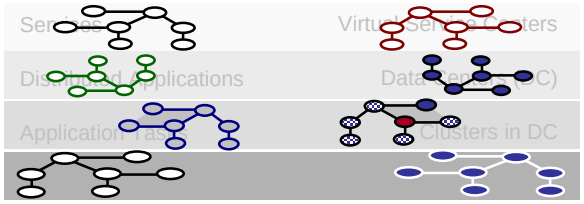

Figure 5: Model descriptions *may* exist for all environments

4.1 Model Descriptions

The abstractions for model descriptions are possible because of the common pattern the behavior of components and environments follows. This pattern is based in queuing theory, and dynamic traffic modeling, enhanced by means to express demands and capacities.

Model descriptions contain following elements [6]:

- to express structure (topology):
  - sets of components represented as *nodes* with or without *memory*,
  - sets of *links* represent relationships between nodes,
  - *nets* as patterns for common network and interaction topologies;

- to express *parameters* of nodes and links as:
  - fixed (static) parameters,
  - variable (dynamic) parameters;
- to express hierarchical structures in models:
  - *sub*-relationships.

Elements used in simulations are:

- *items* representing flowing data or tasks through the model – items can be induced and distributed at nodes in models, and
- *workload* characteristics.

Models are described in a Lisp-like input language as shown in the example fragment below. XML representations can be generated and are used for external access and processing.

```
Top := ( subs (sub $IntraNet_ sub $DSU_ )
        links ( link (from IT to DSU.EDS.E band 1250.0 )
                link (from DSU.EDS.E to IT band 1250.0 ) )
        name DEM
        )
IntraNet_ := (type Clients name IT
        memory (type $Queueing size $MemSize )
        request_shares (s.A 50 s.B 25 s.C 15 s.D 10
                        s.F 0 s.S 0 s.M 0 )
        )
DSU_ := (type Cluster name DSU
        subs ( sub $EdgeNodes_ sub $ApplicationNodes_
               sub $StorageNodes_ sub $AdminNodes_ )
        partition ( type GAPartition max_services 2 )
        net ( type Star name S band 1250.0 with_parent 1 )
        )
ApplicationNodes_ := ( type Cluster name APPS
        memory (type $Queueing size $MemSize )
        subs (sub $App4137_ sub $App4138_)
        net (type Star band 12500.0 with_parent 1 )
        services { s.A s.B s.C s.D }
        partition (type GAPartition max_services 2 )
        )
App4137_ := (type ApplicationNode name A38
        memory (type $Queueing size $MemSize )
        delay $AppNodeDel
        )
App4138_ := (type ApplicationNode name A38
        memory (type $Queueing size $MemSize )
        delay $AppNodeDel
        )
```

Figure 6: Illustration of an internal model description

Descriptions are interpreted by a model interpreter. The model interpreter must know all types of elements referred to in the model. It is currently not possible to introduce new element types in model descriptions themselves. Examples for element types in Figure 6 are: Clients, Cluster or ApplicationNode. New element types must be implemented in C++ using the CSL library and linked into the model interpreters. [6] explains in detail how this is done. The System Factory framework offers a set of predefined element types covering the most common cases.

Nodes represent locations where requests and data are passed through. In accordance to the modeled environment, nodes may represent machines with certain processing capacity or application tasks such as a database capable of handling a certain amount of transaction load. Nodes may also represent whole data centers with certain processing capacity available for certain application services. All these examples show that the model notation is generic. It depends on the interpretation what a model represents.

Similarly, links may represent physical network connections among machines, they may represent bandwidth installed between data centers of a corporate network or also communication activity between two applications. It again depends on the interpretation and the context what model elements represent.

We combine this generality with a systematic way of classifying components of a real system into layers according to granularity.

4.2 Generalized Description of Capacity and Demand

Similar notions of model descriptions have long been used in modeling and simulation. However, they are not sufficient for our purpose of representing demands and capacities in convenient ways. We thus have extended this notion by a generic notion of expressing demand and capacity.

Since nodes primarily represent elements of demands and capacities, we represent each node in various vectors or matrices describing demand and capacity for certain parameters (explained below). Links represent transmission capacity and demand between nodes in a matrix. In their entirety, these data structures represent a consolidated summary of demand and capacity in a compact format. The data structures are seen as extensions to the model descriptions discussed above and forming together with them the description of a *model environment*.

Theory of Relativity: Normalizing Demands and Capacities

In order to enable correlations between different model environments, demand and capacity parameters are represented in a normalized notion. Normalized means not in terms of absolute measures such as total amounts of memory installed or processing capacity, but relative to each other by relating them to an arbitrarily chosen base unit within each modeling environment. For example, instead of characterizing one machine node with a processing capacity of 4 CPU's of type IA64 and another machine with 16 CPU's of type PA850, we chose one of the platforms and assign it the processing capacity of 1.0 (=base unit). The capacity of the other platform is then expressed relatively to this base unit. Assuming that 1 IA64 CPU is capable of the processing of 2 CPU's of type PA850, and the 4 CPU IA64 machine is chosen as base unit, the resulting processing capacities for both machines would be: 1.0 for the IA64 machine and 2.0 for the PA850 machine. The term 'processing capacity' of a machine, however, does not only represent the type and number of CPU's. It is a characterization of the overall machine capacity in one parameter summarizing all different parameters of clock speed, cache sizes, RAM etc. Similarly, demands can be expressed relatively to each other in the same modeling environment. This principle is generally applicable throughout all layers and all modeling environments.

Optimization of the matching of demands with capacities does not depend on the absolute values of capacities or demands. Solutions are the same with absolute and normalized parameters. This enables us using normalized parameters and their advantages.

In order to correlate two different model environments, the two parameters sets for demands and capacities have to be set into proper relation by so-called correlation factors which translate demands and capacities expressed relative to one environment into demands and capacities of another environment by multiplying with these factors.

Determining Normalized Parameters

Absolute capacity parameters as specified for individual machines or demand measured in systems can be used to derive normalized values. In particular for software at higher system layers it is difficult to extract these values. Estimates can then be used as approximations as, for instance, described in initial system specifications. Parameters may even be guessed or defaulted to 1.0 when no information is available at all. The System Factory environment provides means to validate and calibrate these parameters by observations in the real system. Realistic parameter sets will then evolve over time. This calibration process is essential for the validation of the model base and an important enhancement of the current System Factory framework compared to its earlier versions [8].

Normalizing demand and capacity parameters enables their comparability and is the primary way to reduce all the diversity found in systems. This diversity primarily prevents formalizing and correlating views from higher system perspectives today. For this reason it is vital to consolidate all the diversity found in systems in a systematic manner as proposed here. It may be argued that too much information considered being important is lost. However, we see in it the primary way to unify diversity and lifting up the focus of consideration towards higher system views. This makes our approach distinct from others. It is complementary to other approaches representing other information and knowledge about systems by other means such as dependencies in e-services management [12].

Formalizing Parameters for Demand and Capacity

Capacity and demand are referred to as rather abstract terms so far. Both must be formalized in order to be compatible and correlatable. Three dimensions are chosen for capacity and demand.

Parameters for capacity are classified into:

- *processing capacity* as consolidated measure for requests, jobs or tasks per time unit,
- *storage capacity* offered by servicing stations,
- *transport capacity* available between servicing stations as consolidated volume units per time.

The same metrics applies to the demand side:

- *processing demand* as requests, jobs, tasks per time unit initiated by an application or service to a servicing station,
- *storage demand* required in servicing stations,
- *transport demand* between applications or services as volume units per time unit or capacity shares.

All parameters for processing, storage and transport capacities and demands are expressed relatively to fictive base units as introduced above. Base units for the parameter categories can freely be chosen for each modeling environment. Individual parameters are then specified as multiples (or fractions) of their base units, and correlation factors are applied when correlating modeling environments.

More formally, each element represented in a model can be represented in data structures assigned to model descriptions at the capacity side by a:

- processing capacity vector -- C_p,
- storage capacity vector -- C_s,
- transport capacity matrix -- C_t

and at the demand side by a:

- processing demand vector -- D_p,
- storage demand vector -- D_s, and a
- transport demand matrix -- D_t.

Each model element with a representation on the capacity or demand side has an entry in respective processing or storage vectors or matrices for expressing transport and communication demands or capacities between each pair of elements.

Model environments with the normalized parameter sets are then used by System Factory's Generic Optimization Framework described later in this report to find approximations of optimal placements of services on servicing stations based on the demand and capacity structures of both environments.

An example of a simple capacity model can be described by two vectors and one matrix for n elements:

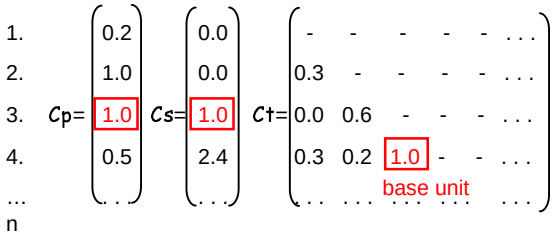

Figure 7: Examples of normalized capacity vectors and matrices for n nodes

In the example, the first element in C_p represents a processing element with a capacity of 0.2 or 20% of the assumed base processing capacity. The following two elements represent exactly the base capacity and the next element 50% of that capacity and so forth. Respectively, storage capacities are shown in the following vector C_s. Transport capacities between node elements are represented in the transfer capacity matrix C_t. Transport capacities between two elements may represent bandwidth installed in a network or communication needs between applications or services depending on the modeled environment.

The same principle applies at the demanding side with respective vectors and matrices for D_p, D_s and D_t. The normalized vectors and matrices for capacities and demands can be extended by further parameters.

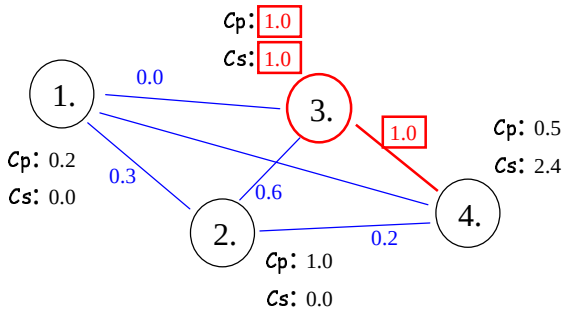

Figure 8: Example of the simple capacity model (according to parameters from Figure 7)

System Organization: Mapping Demand onto Capacity

The goal besides representing and calibrating the information about the behavior of systems from higher perspectives is also optimizing overall arrangements in systems we refer to as *system organization*.

System organization is formalized and reduced to finding optimal mappings of demands onto capacities based on model descriptions. In a top down view, globally distributed services are resolved into participating distributed applications representing certain demands to be allocated in distributed data centers. The model description of an application task represents this demand. The sum of all demands of all application tasks then represents the cumulative demand to be distributed across the data centers. The counterpart models of an infrastructure of distributed data centers represent the capacities to absorb the demands described in the model environments of the application tasks. Based on this information, an optimal mapping or allocation of application tasks in the infrastructure of data centers can be determined by System Factory's Generic Optimization Framework. Optimization policies are represented by configurable cost functions used to evaluate approximated solutions.

Solutions then can be further evaluated by simulations in the System Factory framework and proposed for realization in the system supporting the first stage of the system life cycle. Other stages are supported as well.

Formalizing Constraints

Pure capacity and demand consideration are not sufficient in reality. There are constraints that have to be

met as well. For example, certain application tasks cannot be allocated in foreign data centers for legal reasons though demand/capacity considerations would favor such a solution. Another matrix is currently used for expressing constraints when correlating a demand with a capacity modeling environment:

- *affinity/repulsion constraint matrix* – C_{AR}
  This matrix represents elements for each correlation from the demand side in one dimension and from the capacity side in the other dimension. Probabilistic values between zero and one express that two elements [demand, capacity] must be associated in any case (value 1.0) or must not be associated (value 0.0). Any other value between 0.0 and 1.0 represents the degree of affinity or repulsion between two elements. The value 0.5 is neutral and assumed as default.

4.3 Correlating Models

Model descriptions are internally represented in the notion shown in Figure 6 for historical reasons. The external representations use XML. Conversion between the two formats is straightforward since both model descriptions follow a tree structure. Hyperlink technology is used to access and correlate models from different locations since we assume that the model base of large-scale distributed systems will be distributed as well. System Factory's model base is thus designed and implemented as a distributed model base accessible through web-technology providing the connectivity. XML is the standard for data representation used in many systems today.

Currently, complete model descriptions are distributable elements. It is planned to use XML hyperlink technology to support modularization of model descriptions as well. It would enable to distribute also individual elements referred to in models and linking them together on demand. Element modules could then be maintained separately or even offered by suppliers offering capacity to customers. Customers could then access offered capacity descriptions from providers and evaluate them within their model environments and eventually purchase capacity from them.

But these are considerations for future enhancements. Another enhancement is extending model descriptions by economic factors and time schedules in addition to current capacity and demand parameter sets.

5 The Framework: System Factory

Our approach entirely relies on models about large-scale system representing modeling environments at various layers of granularity. Following questions occur:

- How to obtain models?
- How to keep model descriptions current?
- How to manage large numbers of model descriptions?
- How to evolve models with changes in the system?
- How to reflect dynamically changing parameters in model descriptions?
- How to correlate model descriptions?
- How to integrate statistics into models?

System Factory provides the framework and an integrated environment for addressing these questions. It is intended for system designers and integrators. It should help them quickly adopting customer environments and using System Factory as a smart configurator to identify and underpin organizational decisions in systems from higher perspectives. They can experiment with various scenarios, analyze them, and elaborate the best solutions before being implemented.

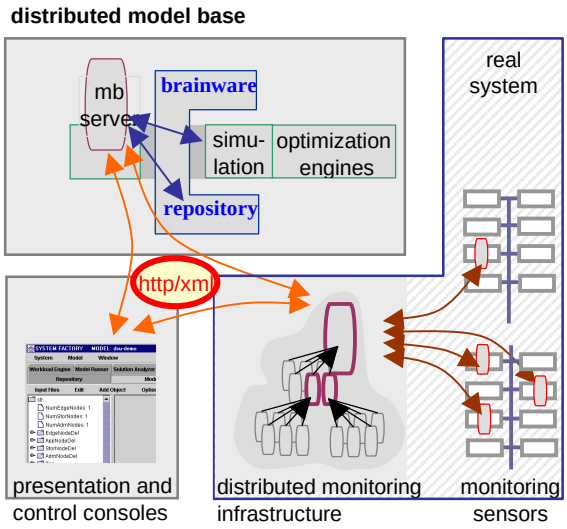

Figure 9: The System Factory Framework

System Factory consists of following main parts:

- *distributed model* base,
- *distributed monitoring* infrastructure, and
- *presentation and control* through consoles.

The framework is inherently distributed with multiple locations for model bases and monitoring. We use web standards for communication (http, email protocols) and external representations (xml) for models, monitoring data, events, and all other data used in the System Factory framework.

The model bases host model descriptions at various locations. Some model bases may also provide further capabilities for optimization and simulation as shown in Figure 9. They will be discussed in more detail later.

The distributed monitoring infrastructure serves the purpose to constantly observe various parameters measured in the real system, process this information and finally extract parameters and assign them to parameter sets of particular model environments. This enables keeping model parameters up to date and also to discover divergence between evaluated and observed parameters.

The presentation and control consoles allow to "look into the system", displaying the various models and parameters. They also allow performing certain control functions such as manipulating model descriptions and correlating model descriptions. The currently Java-based GUI will be replaced by using regular web browsers and advanced XML technology (XVG [13]) for presentation. Examples of the current presentation are shown in the Figures 14 to 16.

5.1 The Distributed Model Base

The D*istributed Model* base forms the heart of the System Factory where all model descriptions are maintained. Models are accessible in their external representation as XML documents by web interfaces.

The Repository

The *Repository* is an intelligent database which stores information obtained and used in the system design and maintenance, namely information about:

- prior and current model descriptions of hardware, software, networks, middleware, applications,
- typical solutions and template models,
- typical workload patterns, traces, benchmarks.

The whole information in the repository is structured into a hierarchy of segments, related to different business patterns and corresponding IT infrastructures. The segments may also differ by levels of detail. XML is used as specification language for the repository as well providing standard and contemporary technology.

Brainware

The overall collection of tools and libraries for decision making for system organization and administration is summarized under the term *brainware* in System Factory.

The Generic Optimization Framework and the Simulation Engine represent current tools of System Factory's brainware and are described in separate sections below.

5.2 The Distributed Monitoring Infrastructure

The Distributed Monitoring Infrastructure provides the link into a real system. Sensors constantly observe certain parameter such as load absorbed by an application by checking their log files or interfacing with existing monitoring infrastructures such as HP OpenView [5]. Parameters are filtered and processed through the distributed monitoring infrastructure and assigned and compared with corresponding parameters in model descriptions. Monitoring serves the purpose of delivering measured data for model validation and calibration, parameter update and the discovery of divergence between evaluated parameters from models and observed behavior in the real system.

The Distributed Monitoring Infrastructure is structured into a hierarchy of domains. Communication between monitoring units is also based on http/xml.

Monitoring Units

Monitoring units are passive in the sense that they do not initiate action in the monitored system. Monitoring should be non-intrusive to the monitored system. Monitoring units perform the task of monitoring the behavior of certain components of a system.

A Monitoring Reference Model [9] defines the elements of a monitoring process performed by a monitoring unit (slightly extended here):

- generating monitoring information:
  - gathering (raw) status data and detecting raw events from observed components;
- local processing of monitoring data:
  - filtering and condensing gathered status data,
  - mapping input status data and event information into primitive ("analog") status variables such as CPU load and eventually deriving discrete state variables from them such as "{low, high}",
  - deriving composite status or state information from primitive information by applying processing functions,

- detecting events from certain status changes or state transitions on primitive or composite status or state variables.

The central data structure containing primitive or composite status and state information about observed components is usually referred to as the Management Information Base (MIB) of the unit.

- collecting statistics of selected monitoring information, and
- dissemination of monitoring data:
  - to inform (usually) higher-ordered units in the monitoring infrastructure about status and state changes and events.

Input to monitoring units is provided by sensors. *Sensors* are responsible for data gathering about observed components. Sensors generate "raw" input data for further processing by monitoring units.

Special kinds of sensors are so-called *Interface Sensors* providing bridges into other monitoring or management systems for the purpose of collecting status information from these sources. SNMP interface sensors could be an example. Interface sensors can also provide hooks into other management systems such as HP OpenView [5].

Another special kind of sensors is used to connect to lower-ordered monitoring units enabling to set up information dissemination topologies (mostly but not necessarily forming hierarchies corresponding to monitoring domains) for monitoring data. Such sensors are called *Connector Sensors*.

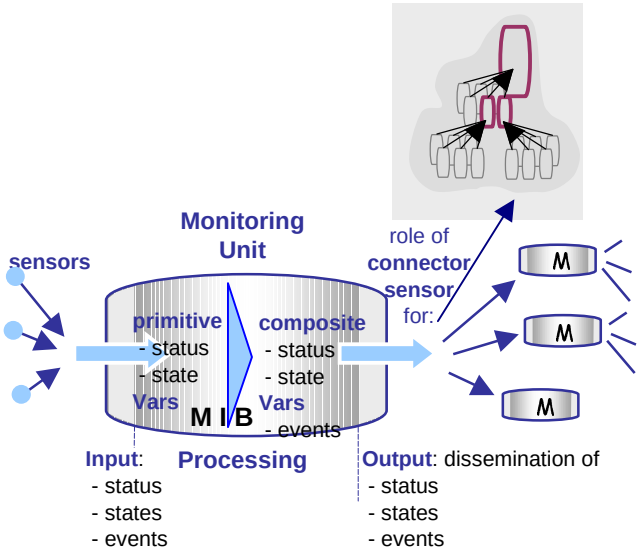

Figure 10: Topology of monitoring units

Sensors' raw data are first filtered and condensed by performing some functions and are then mapped into (primitive) status variables in the MIB. According to changes in primitive status variables, composite status variables may be updated as well. Composite status variables allow representing more abstract or higher-level status information.

Status variables represent discretized values of observed analog parameters such as an observed load "measured between 0% and 100%". Functions transform primitive into composite status variables. Discrete state variables may be used as well. Discrete state variables are not defined by ranges of numeric types but by explicit sets, for instance, a state variable may represent load as "{low, normal, high}". Functions map status variables into state variables. Mapping status into state variables provides further filtering and abstraction. State machines then are applied to perform transitions upon state variables and to derive composite states.

An important part of processing monitoring data is event detection. Events are defined by event conditions. Event conditions are represented by boolean functions depending on a set of status or state variables maintained in the MIB of a monitoring unit. The boolean function is evaluated on any change of any of the dependent status or state variables. If the function with the event condition evaluates 'true', it is said that an event has occurred.

The output of an monitoring unit is monitoring data:

- primitive or complex status or state information,
- changes in status or state variables, and
- events based on event conditions.

Each of the data will be locally time-stamped when issued to other units. Monitoring units may collect certain monitoring data persistently over time allowing statistical evaluations (traces, use patterns, workloads, traffic, error behavior and more). Statistics are collected in databases separately from monitoring units.

Subscribe-mechanisms provide higher flexibility then fixed reporting paths for the dissemination of monitoring data. It enables dynamic changes and allows replication of monitoring paths for improved robustness and availability of the monitoring system.

Other characteristics are communication patterns how and when monitoring data will be sent to subscribers:

- *scheduled push* – driven by configurable schedules (e.g., every 100ms, every Monday, 10 o'clock, etc.),
- *event-driven push* – at the occurrence of an event,
- *subscriber poll* – on request of subscribers.

5.3 The Generic Optimization Framework

The *Generic Optimization Framework* (GOF) is an engine performing the described mappings of application demands onto capacities provided by their infrastructures. The Generic Optimization Framework is part of System Factory's model base used to improve the overall design and organization of global-scale enterprise and e-services systems. GOF allows solving optimization problems falling into the class of general arrangement or mapping problems typically classified as NP-hard so that approximations are applied in practice. We currently use a Genetics Algorithm to approximate solutions [7]. Problems are characterized by finding mappings of one set A into another set B by meeting some optimality criteria and taking constraints into account.

In the context of System Factory, mappings refer to distributing the introduced notions of abstracted 'services' among 'servers', optimized under goals such as reducing the overall traffic in the system or balancing load. Services are represented by a vector A. Further input data describe services' demands for processing, storage and communication in the parameter set for A. Servicing stations or servers are described in vector B with respective processing, storage and transport capacities described in B's parameter set. The result of an optimization is a mapping matrix containing a solution of services assigned to servicing stations.

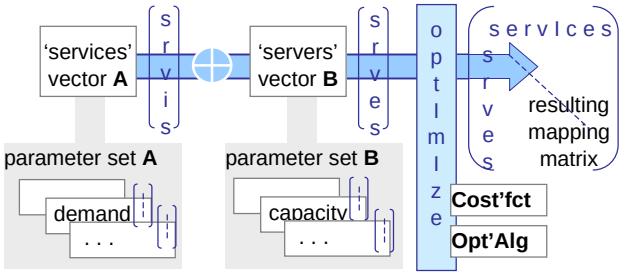

Figure 11: Principle for optimizations

The figure shows the general principle with data structures represented as vectors and matrices. Further parameters may be included in parameter sets as well. The figure does not show the affinity/repulsion constraint matrix – C_{AR} – taken into account for generating and evaluating solutions (if present).

An optimization is performed by a configurable optimization algorithm based on a configurable cost function representing the optimization goal or policy. Input parameters and output results are represented in a normalized (generic) form. Backward mappings then translate results contained in the normalized result matrix back into the application space based on separately provided translation descriptions.

Generic Optimizations

Since optimizations follow the same pattern for all layers, and all input and output data are normalized for internal processing, the same generic optimization algorithms, policies and principles can be applied.

Optimization algorithms follow the same iterative pattern for finding approximations:

1. generate a possible solution;
2. evaluate the solution according to constraints and an optimization goal or policy;
3. a "better" solution returns a lower cost value from the evaluation by a cost function;
4. if the evaluated solution is better than prior solutions, replace the worst solution in the solution pool with the generated solution;
5. repeat the cycle until some termination criteria applies.

Figure 12: General pattern of optimization algorithms

The GOF provides the architectural framework for the described optimizations. Its main functions are parameter normalization, performing optimizations based on configurable optimization algorithms and cost functions representing optimization policies. Results are translated from their normalized representations back into the application space by the Solution Mapper (Figure 13).

Normalization in GOF refers to applying the correlation factors to two model environments in order to make them comparable. A better name probably should have been found for GOF's normalizations to distinguish them from the normalizations applied in System Factory's models for abstracting and consolidating the diversity of absolute parameters.

GOF is based on approximations for finding optima. Different algorithms exist for approximations. For small solution spaces, complete enumeration and evaluations by a cost function may be applied. For larger dimensions of models, subsets of the solutions space may be evaluated. As we experienced, Genetics Algorithms provide a fast and good approach for finding optima in larger-scale systems up to few hundred nodes [7]. More algorithms exist. All these algorithms are based on the same pattern of generating and evaluating solutions. GOF allows

incorporating many of such algorithms into a general framework as configurable entities.

Similarly, cost functions evaluate a particular solution according to optimization goals. They also follow the same pattern: better solutions according to the goals are evaluated with better cost values. This allows generalizing cost functions and mapping them into the general framework as configurable entities as well.

The set of optimization algorithms and cost functions is orthogonal meaning that each cost function can ideally be combined with any optimization algorithm.

Different optimization goals will require different sets of input parameter used in cost functions. Each cost function hence also requires its own normalizers for respective parameter input sets.

Architecture of the Generic Optimization Framework

The following figure shows the architecture of the Generic Optimization Framework:

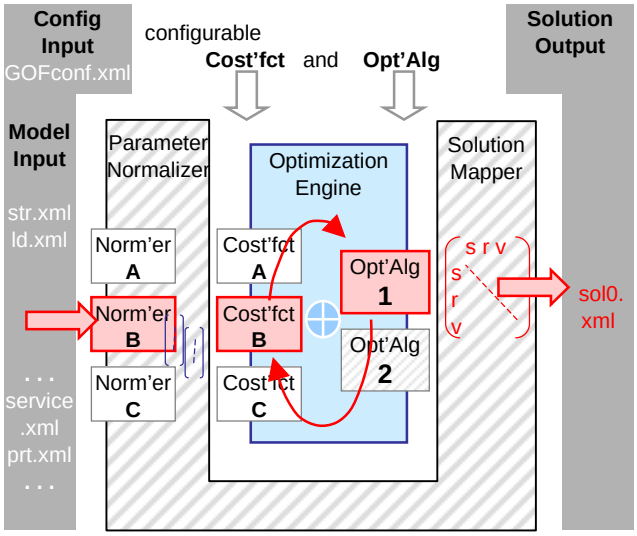

Figure 13: General Optimization Framework Architecture

Optimization algorithms and cost functions are provided as configurable entities to GOF. Configuration input data determine which algorithm in combination with which cost function will be used for a particular optimization.

GOF will be hosted on dedicated machines called GOF optimization engines. Conceptually they belong to System Factory's model bases. All input and configuration parameters will be provided as a set of XML documents representing one optimization task. GOF is accessible through web http/xml interfaces.

5.4 The Simulation Engine

The simulation engine allows further validation of proposed optimization solutions or provided scenarios. Simulations are based on the Communicating Structures Library (CSL), a C++ simulation library developed at HP Labs [6]. Various workloads can be described and simulated. 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. CSL uses C++/CSIM, a commercial process-oriented discrete-event simulation package.

CSL nodes can be assigned processes that generate items, receive or send items from or to other nodes. They can also transform items. A node with assigned processes becomes a simulation node and is able to model active components of systems. Processes exchange information with other processes 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 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 waits on the event if the event has not occurred yet.

CSL allow us to construct also queuing networks. A CSL node and a CSL net (comprised by links) 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 the number of servers in the node. The input data for the queuing model are an inter-arrival rate and a mean service time. The model returns the average waiting time, 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 node:

- the arrival rate from outside the network,
- the probability that an item moves from node A to B,
- the service time and 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.

6 Presentation and Control

Presentations are currently based on model descriptions that only contain topological structure and behavioral parameters introduced in the section about model descriptions. No graphical presentation information is contained in model descriptions. The current lack of graphical presentation information only allows abstract presentations of models.

XML technology enables keeping graphical presentation information separate from model content. The mixture of model content and presentation content in model descriptions is a major drawback in other modeling packages such as MATLAB/SIMULINK.

SVG [13] is an emerging XML standard for vector graphics enabling customized presentations of models with keeping model content information separate from graphical presentation information, both represented in XML, and both are merged together in a XVG-enabled display device such as recent web browsers. This is a planned enhancement to enable the customization of presentations.

6.1 System Factory User Interface

The System Factory User Interface is currently implemented as Java-GUI. It presents several functions to the user organized as so-called *shops*.

To make System Factory not just 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.

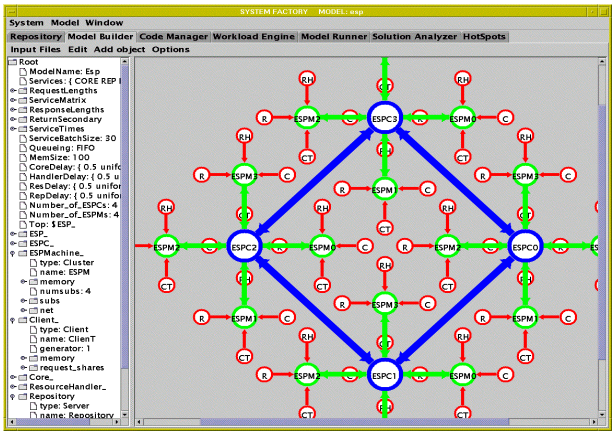

Figure 14: Java-based GUI for System Factory

Figure 14 shows the *Model Builder* shop. The left panel contains the textual specification of an e-services model, which is constructed by a user or is customized using a template model from the repository. The right panel displays that fragment of the model communication structure in the mentioned abstract form.

Several shops categorize the System Factory User Interface:

- *Workload Engine* provides the capability to analyze traces received from the repository or from the so-called Measureware shop; or synthesizing artificial workload and feeds it into the Model Runner.
- *Model Builder* allows describing system topologies, system parameters, as well as variations in topologies and parameters in both textual and graphical form.

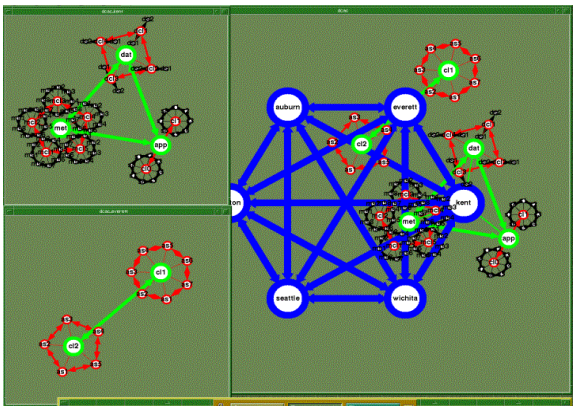

Figure 15: Visualization in the Solution Analyzer

- *Measureware* accesses resource and performance data from the monitoring infrastructure. The collected data may be stored in the repository or fed into models.
- *Model Runner* actually prepares and runs simulation experiments according to the system integrator's objectives.
- *Solution Analyzer* analyzes simulation results and presents them in a graphical form as shown in Figure 15.
- *Solution Synthesizer* looks for better solutions in the solution space typically available after modeling of complex systems with multiple optimization criteria.
- *Management Shop* implements the model-based tasks of system organization and management. The shop provides access to the system manager (or a hierarchy of distributed system managers) that

observe system parameters and behavior through the monitoring infrastructure. The managers react on reported event conditions and act by notifications though the GUI if applicable and/or by automatically updating model parameters.

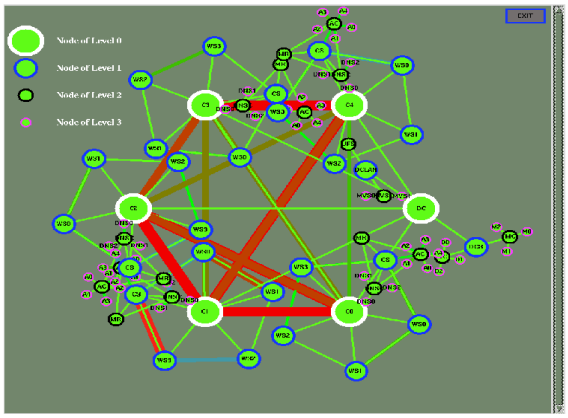

Figure 16: Visualization in Simulation Run

7 Case Studies

Two case studies have been carried out with an earlier version of the System Factory framework to gather experience with large customer environments. This earlier version basically consisted of the modeling and simulation environment of System Factory. It did not have the technology we have recently developed and described in this paper: the homogeneous approach of matching demand with capacity at different layers of granularity, the Distributed Model Base, the Generic Optimization Framework and the Distributed Monitoring Infrastructure enabling automatic model calibration and validation. These technologies were considered as needed after the experiences made with the earlier version of the System Factory.

7.1 Case Study 1

The case study considered a worldwide distributed IT system for a global transportation company (FedEx). This environment had been modeled and analyzed using CSL modeling.

Considered areas were:

- tracing package delivery paths (hundreds of millions of transactions per day),
- processing for billing,
- customer services for web access,
- common internal enterprise business applications.

The overall IT infrastructure was represented as a three-level hierarchical network of a system 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.

Requirements were “almost real-time” computing and cost effectiveness by fulfilling the overall processing tasks with an efficient amount of systems.

Information was 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. Two alternative choices were investigated: point-to-point communication and dispatch by data brokers (or information hubs). The first case represents a web of point-to-point links, which were hard-coded for specific platforms, applications and data formats. In the second case, so-called data brokers represented shared message routing hubs. Brokers maintained the tables of subscribers for each type of the messages and forwarded messages to the subscribers. The task was to evaluate the two choices.

The constructed CSL model contained those system features that influenced the message traffic and were important for satisfying the global system requirements. The model helped to identify bottlenecks and the system sensitivity to changing parameters such as the number of centers, bandwidth in local and global networks, various kinds of message packaging, etc.

The sensitivity and utilization analysis included:

- system response times on bursty workload,
- system scalability with an increasing number of participants,
- data broker overhead,
- various data broker topologies.

The main result of the project validation was the reduction of the proposed three-level system architecture to a two-level architecture. This is an example of an organizational change or decision at the very top-level system perspective. The CSL analysis of the traffic in the system had shown that when the functions of the second-level data centers were moved to the top-level data centers and the lower-level operation centers, the global traffic became less congested, response times were improving while still satisfying the basic system requirements. The overall cost could be, of course, dramatically reduced. The general approach of this modeling may serve as a template model for other company-wide IT infrastructure of global delivery companies and stored in the repository.

7.2 Case Study 2

A complex case study was the validation of the enterprise-wide IT infrastructure of the Boeing aircraft corporation. The main task was to predict the system scalability with the number of workstations increasing in the range from several 1000s to several 10,000s.

The system had four tiers: workstations, application servers, method servers, and database servers. Method servers provided support for the product data management programs accessing and managing the data in database servers. The traffic of requests was propagated from the first tier to the last. The response traffic was passed back. Requests had different priorities, and responses had varying lengths. The request-response traffic could be bursty. A generic model was constructed that actually became the basis of the *Systems of Servers* modeling template as part of System Factory's repository.

8 Related Work and Summary

Investigations as described in the case studies have widely been carried out for large IT environments as for example shown in [11]. Models are usually specifically constructed for particular investigations. It is hard to validate those models. This leads us to the perception to accompany complex operating IT infrastructures with an integrated, widely self-maintained, validating and calibrating model infrastructure as a meta-system. The distributed model base realistically reflects the system's landscape and behavior in order to integrate and simplify investigations. We thus step beyond what is being addressed in traditional system modeling.

In regard to system management systems, our focus is on higher system perspectives then lower-level component views usually covered by those systems.
AI-based approaches have been investigated for automating system management tasks as well. They rely on (too) detailed and complete "logical" information about systems, which is hard to obtain and maintain in large-scale contexts. We prefer statistical, "behavioral" information uniformly represented in models.

Another new approach we propose is in the consolidation and reduction of the diversity found in real systems into few, basic parameters for demand and capacity enabling us to integrate the various layers of systems, including the layers of software systems, in a uniform manner in continuation of the approach investigated in [4].

We believe that by our approach a better understanding of systems will enable decision support, automation and optimization for system management and system organization from higher perspectives to systems.

References

[1] Distributed Management Task Force (DMTF): *Common Information Model (CIM) Specification*, http://www.dmtf.org/spec/cim_spec_v22, June 1999.

[2] DMTF (Distributed Management Task Force, Inc.): *Specification for the Representation of CIM in XML*, Version 2.0, http://www.dmtf.org/download/spec/xmls/CIM_XML_Mapping20.htm, July 20th, 1999.

[3] Graupner, S., Kotov, V., Trinks, H.: *Distributed Service Management with System Factory*, HP Labs Technical Report, HPL-2000-152, November 17th, 2000.

[4] Graupner, S.: *A homogeneous Architecture for Applications and Operating Systems*, Ph.D. Thesis, Chemnitz University of Technology, Germany, 182 pages, Shaker Verlag, ISBN 3-8265-3261-9, November 1997.

[5] Hewlett-Packard: http://www.openview.hp.com.

[6] Kotov, V., Rockicki, T. M., Cherkasova, L.: *CSL: Communicating Structures Library for System Modeling and Analysis*, HP Labs Technical Report, HPL-98-118, June 1998.

[7] Kotov, V., Trinks, H.: *Optimization of E-Service Solutions with the Systems of Servers Library*, In Proceedings of MASCOTS 2000, pp. 575-582, August 29th – September 1st, San Francisco, September 2000.

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

[9] Mansuri-Samani, M.: *Monitoring of Distributed Systems*, Ph.D. Thesis, 165 pages, University of London, UK, December 1995.

[10] Martin-Flatin, J.P., Znaty, S., Hubaux, J.P.: *A Survey of Distributed Enterprise Network and Systems Management Paradigms*, Journal of Network and Systems Management, Special Issue on Enterprise Network Systems Management, Vol. 7, No. 1, March 1999.

[11] Oleson, V., Eisenhaur, G., Pu, C., Schwan, K., Plale, B., Amin, D.: *Operational Information Systems – An Example from the Airline Industry*, First Workshop on Industrial Experiences with Systems Software (WIESS 2000), San Diego, October 22nd, 2000.

[12] Sahai, A., Machiraju, V., Wurster, K.: *Managing Next Generation E-Services*, Hewlett-Packard Laboratories Technical Report HPL-2000-120, September 2000.

[13] W3C: Scalable Vector Graphics (SVG) Overview: http://www.w3c.org/Graphics/SVG, created in February 1999.