36. Generic Core Life Cycle and Conceptual Architecture for the Development of Collaborative Systems

Tad Gonsalves and Kiyoshi Itoh
Information Systems Engineering Laboratory, Faculty of Science & Technology, Sophia University, Tokyo, Japan
Email: {t-gonsal, itohkiyo}@sophia.ac.jp

In the conventional system development life cycle (SDLC), the system performance evaluation phase comes after the implementation phase. Our strategy is to project system performance estimate at the requirement analysis and design phase itself, much before the implementation phase. To achieve this objective, we propose a technology-neutral integrated environment for the core life cycle of system development. This core life cycle consists of three phases: system modelling, performance evaluation and performance improvement.

1. INTRODUCTION

Testing and evaluating system performance is an important stage in the development of a system. However, it is often ignored due to lack of time, tools or both. While designing systems, "designers are (blindly) optimistic that performance problems – if they arise – can be easily overcome" (Cooling, J, 2003). The designers test the performance of the system after the completion of design and implementation stages and then try to remedy the problems. The main difficulty with this 'reactive approach', Cooling states, is that problems are not predicted, only discovered. It is the concern of this study to *predict* the operational problems and to suggest a viable solution while designing the systems. The significance of performance design lies in the fact that the performance requirements and performance characteristics are already incorporated in the system at the design stage of the system. The system designers project an estimate of the system performance, as it were, at the requirement analysis stage itself of the system development life cycle.

In this paper we propose a core life cycle and an integrated environment for system development. The core life cycle consists of three phases - modelling, performance evaluation and performance improvement. System performance is evaluated by simulation and an initial improvement plan is suggested by the expert system. This improvement plan is incorporated in the system model and the performance evaluation simulation cycle is repeated. Thus, the originally planned system and its operation could, in principle, be repeated through a number of cycles in the integrated environment, till a refined system is obtained. Further, the core life cycle can be incorporated in the total system development life cycle.

Our target systems are collaborative systems. When multiple organizations work on a joint project or come together to share resources to enhance their business prospects, they constitute a collaborative system. The conceptual model of the

system depicts the interrelationships between actors, tasks and collaborative activity in the system. Corresponding to these, actor sufficiency, task mobility and activity efficiency are chosen as performance indicators of the system. Performance improvement is by an expert system driven by qualitative rules. We have developed a systematic methodology for the conceptual core life cycle and indicated an implementation scheme for practical collaborative systems.

This paper is organized as follows. Section 2 introduces the salient features of collaborative systems which may be exploited in modelling the system and in evaluating and improving its performance. Section 3 describes the three integrated phases of the core life cycle. Section 4 deals with the conceptual architecture for the core life cycle and Section 5 describes in brief our attempts in implementing the core life cycle for real-life collaborative systems.

2. SALIENT FEATURES OF COLLABORATIVE SYSTEMS

When multiple organizations work on a joint project or come together to share resources to enhance their business prospects, they constitute a collaborative system. Business firms collaborating to enhance their business prospects, doctors and nurses in a clinic offering medical service to patients, teachers and staff members in a school offering educational service to students, manager and tellers in a bank offering financial service to customers, etc., are all examples of collaborative systems. Another special class of systems engaged in collaboration is collaborative engineering systems. The goal of these systems is to do engineering (or to provide 'engineering services') through collaboration.

Collaborative systems have several distinct features that may be exploited in the modelling, performance evaluation and performance improvement of the systems. In this section, we describe three salient features of collaborative systems that may be used in the different phases of system development.

2.1 Request-Perform Activity

Each collaborative activity in a collaborative system may be looked upon as a request-perform activity. Collaborative activity begins when the requesting collaborator requests the performing collaborator to perform the activity. The overall workflow in the collaborative system can then be viewed as a series of interrelated request-perform activities. Some request-perform activities are sequentially linked while others are concurrent; still others need to synchronize with one another.

The request-perform activity property can be used to adequately represent the collaborative system under consideration. The topology of all request-perform activities with their inter-relationships (sequential, merging, diverging, concurrent, synchronizing, etc.) can serve as a convenient system model.

2.2 Client-Server Systems

A client is anything that places a request for service and a server is anything that offers service in response to the request. In collaborative systems, the collaborators act as servers, offering service to the clients. Clients, in general, could be customers, orders, material for production, etc., that enter the system seeking service. Sometimes a given server may perform some part of the collaborative activity and request another server for the remaining part of the service; in other words, servers could become clients of other servers.

Waiting-line analysis, also known as queueing theory, deals with the computational aspects of the server in the client-server setup (Figure 1). Each collaborative activity may be represented by the queueing server and real-life collaborative systems may be modelled as a combination of several appropriate servers.

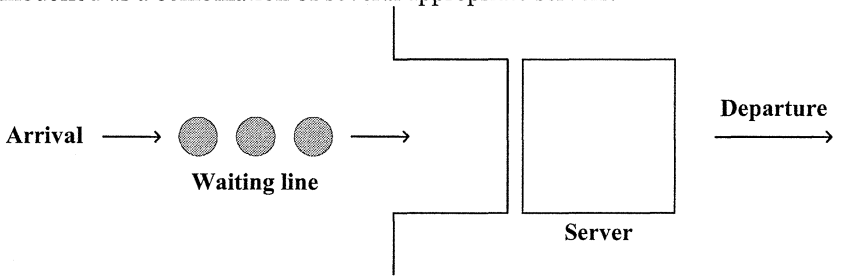

Figure 1: Model of a single queueing system

2.3 Discrete-Event Systems

A system is defined to be discrete-event if the phenomenon of interest changes value or state at discrete moments of time, as opposed to the continuous systems in which the phenomenon of interest changes value or state continuously with time (Banks, J, Carson, II, J.S, 1984). Collaborative systems are discrete-event systems because the events take place in discrete steps of time. An event is an occurrence that changes the state of the system. The beginning of service and the end of service are typical events in collaborative systems. In a clinic, for instance, a patient arriving for service, joining in the queue if the physician is busy, starting treatment and ending treatment are significant events that change the state of the system.

Both stochastic and deterministic systems can be simulated by using the discrete-event simulation approach (Ross, S.M, 1990). Discrete-event simulation is centred around the concept of event. This is because the only significant thing that contributes to the time-evolution of the system is the series of events. Without the series of events occurring in the course of the history of the system, the system would be static. Discrete-event simulation is just the right approach to simulate the performance of the discrete-event collaborative system.

3. CORE LIFE CYCLE

Our ultimate aim is to estimate the performance of the system at the requirement phase of the SDLC. To achieve this aim, we create a generic core life cycle consisting of just three phases - system modelling, performance evaluation and performance improvement. Each of these phases is described in detail in the ensuing sub-sections. From the planned system, a rough model is made in the beginning. The modelled system is simulated to yield its performance. The performance results are evaluated and suggestions for improvement are provided by an expert system. The operation improvements are incorporated into the model of the system and the entire cycle is repeated, yielding a refined system after each cycle. Furthermore, we make this approach integrated so that each phase, if not automatically, at least semi-automatically leads to the next phase.

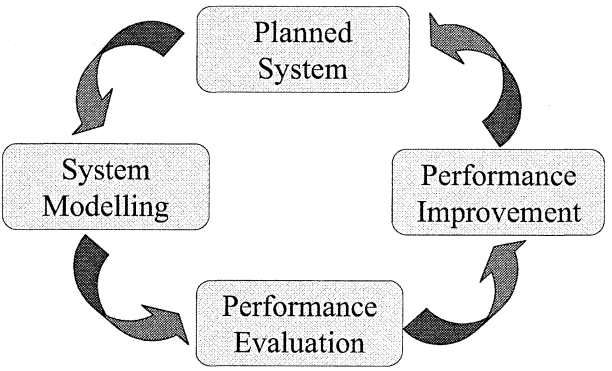

Figure 2: Core life cycle

3.1 Modelling

Model is an abstract concept that stands for the system. It is not the system itself, but something that represents the real system. Models should be made simple enough so that they can be simulated and improved. The model designer should not try to include too many details in the model, because that would make the model cumbersome and unwieldy. On the other hand, the designer should not make oversimplifications, because then the model will be far from the real-system it is supposed to represent.

The type of model to be chosen will depend on the system at hand. The model of a collaborative system should have the ability to account for the prominent features of collaboration. Any form of collaboration necessarily implies the presence of 'actors' (collaborators), 'activities' and 'objects upon which the action is performed' (tasks). Each of these constituent elements of collaborative systems is briefly described below.

Actors

In collaborative systems the collaborators are the main actors. They keep the system going by carrying on the collaborative work. They receive tasks and process them and make the system realize its ultimate goal. Actors, in practice, could be personnel or machines or a combination of both. The interaction among the actors is what essentially determines the nature of the collaborative system. The system model should have the capacity to grasp the prominent interactions among the collaborators.

Tasks

Tasks are processed by collaborators in the collaborative system. There could be a variety of inter-related tasks flowing in a variety of ways in the system. In designing the system model, the modeler should be able to decipher the different tasks and elaborately analyze their flow in the system. The flow analysis should lead to a well-defined flow-control mechanism in the model.

Activity

The work done on the task, (or the processing of the tasks) by the actors, is called activity. Collaboration consists of a series of activities that are linked to one another

in a logical way. The collaborative activities present in a collaborative system are numerous and of a diverse nature. A sound model should be able to enumerate all the activities systematically, group them and arrange them in the model layout so as to give a comprehensive view of the entire system.

3.2 Performance Evaluation

Performance evaluation is more an art than a science. There is no hard and fast rule for performance analysis. It varies from developer to developer. Further, it may vary from system to system. Performance evaluation, to a great extent, will depend on the features of the system chosen by the manager/designer. In this section, we present some of the basic performance indicators that are of interest when faced with the task of performance evaluation of a collaborative system.

Since we have singled out actors, tasks and activities as the basic constituent elements of our collaborative system model, it follows that the performance indicators of the system operation should be constructed around these three basic elements. We choose the following three key performance indicators.

Actor sufficiency

The collaborators in a collaborative system are the actors in that system. They receive requests for service, process the requests and forward the requests (tasks) to subsequent servers in the system. When the actors at a given server are insufficient, service time tends to prolong and the utilization of the server increases. With the increase in service time, the queues in front of the servers grow in size, making the system operation unstable and leading to greater customer dissatisfaction. However, increasing the number of actors in nearly all cases leads to an increase in cost. Actor sufficiency, therefore, is a trade-off between operation cost and customer satisfaction.

Task mobility

Task mobility refers to the frequency of tasks' entry into the system and their flow from server to sever in the system. Another term for task mobility is workload and consists of service requests to the system. In the analytic approach, task mobility is usually represented by different probability distributions. We define task mobility as

$$\lambda = \text{Number of tasks / Time}$$

Task mobility is an important factor influencing the performance of the system. If the task mobility is low, system utilization is low, implying that the system resources are underutilized. However, system utilization rapidly increases with the increase in task mobility and soon approaches unity. Therefore, task mobility should be maintained at an appropriate level so as to maintain the system utilization in the optimum range of operation.

Activity efficiency

The utilization of the server is given by

$$\rho = \lambda / \mu$$

where μ is the service rate.

The utilization of the server, ρ, represents the fraction of total operation time the server is kept busy. This ρ, sometimes referred to as the traffic intensity, is also a measure of the efficiency of collaborative activity. System managers generally tend to drive the system to its maximum utilization. However, it is a fact of observation that the response time tends to infinity as utilization approaches unity. Storage is another problem that presents itself in the face of rapidly rising queues because of high utilization. From the failure-to-safety aspect, experts' heuristics suggest that the server utilization may not exceed 0.7 (Itoh, K, Honiden, S, *et al.*, 1990).

3.3 Performance Improvement

3.3.1 Qualitative Rules for Performance Improvement

Collaborative systems are usually large complex systems. It may not be possible to grasp all the variables that are in the system and the diverse interactions among them. Thus, quite a few complex collaborative systems could be systems with incomplete knowledge. Modelling, performance evaluation and performance improvement of such systems could be done by way of Qualitative Reasoning (QR). QR is an AI discipline that bases its reasoning on the nature and behaviour of the components that make up the system (Kuipers, J, 1994). A set of qualitative rules presented below can be established by accumulating knowledge of experts in the domain.

Table 1- Qualitative Rules for Performance Improvement

Performance Indicators	State	Number of actors (N)	Rate of activity (μ)	Rate of task mobility (λ)
Actor Sufficiency	Scarce	↑	↑	↓
	Normal	○	○	○
	Excess	↓	↓	↑
Task Mobility	Scarce	↓	↓	↑
	Non-uniform	○	○	↑/↓
	Excess	↑	↑	↓
Activity Efficiency	Minimal	↓	↓	↑
	Normal	○	○	○
	Critical	↑	↑	↓

(Notation ↑: Increase parameter; ↓: Decrease parameter; ○: Do not change parameter)

The above principles could be further elaborated and structured as IF-THEN production rules, which can then be used in the construction of the performance-improvement knowledge-based system.

3.3.2 Meta-Rules for Performance Improvement

In addition to the above practical rules, one could envisage the following meta-rules that would be needed for the application of the above rules. The meta-rules act as guiding principles for the application of the performance improvement rules.

Availability

The parameters (number of actors in a given activity, duration of the given activity, task mobility, activity efficiency, etc.) that could, in principle, be changed so as to improve the performance of the system, satisfy the desirability condition or the availability condition. All these parameters are included in the available set. This is the largest set shown in Figure 3.

Feasibility

However, the values of some but not all the members of the available set can be changed. The system operation requirements and constraints are such that certain parameters cannot be changed. In other words, changes in these parameters are not feasible under the given conditions. Only the parameters that may be changed are included in the feasible set. Further, there may be a limit to the range of changes that are feasible for a given parameter. The feasibility conditions are normally laid down at the requirement analysis stage of the system development. The feasibility knowledge is therefore another important factor that needs to be included in the performance improvement knowledge-based system.

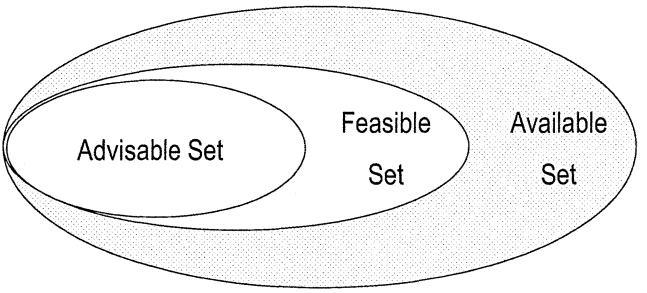

Figure 3. Available, Feasible & Advisable Sets

Advisability

In resolving bottlenecks, more important than the feasibility condition is the advisability condition. At times, it may be feasible to change a parameter, but the knowledge-based system (KBS) may judge that it is not advisable to change it for fear of exerting bad influence on other sections of the system. The advisable set, therefore, is only a subset of the feasible set. It contains only a handful of parameters which satisfy all the three conditions of availability, feasibility and advisability in resolving bottlenecks locally, while maintaining the global stability in system operation. If the changes made in one section of the system will end up in creating fresh bottlenecks or worsen the existing ones in the other section, then the *advisability* condition will prompt the KBS to issue a warning to the user. Advisability means coming up with a sound strategy for resolving bottlenecks such that changes made in the network are minimum.

4. CONCEPTUAL ARCHITECTURE FOR THE CORE LIFE CYCLE

The core life cycle of system development seeks to integrate the three phases of modelling, performance evaluation and performance improvement. The model of the collaborative system is typically a network of request-perform activities inter-related according to the logic of the collaborative system workflow. The model could be multi-layered, with each layer depicting in detail some aspect of collaboration. At the deepest level, there should be a representation of the network structure of the system.

Through an inter-conversion scheme, the network structure in the model could be imported into the Performance Evaluator (PE) and KBS (Figure 4). The qualitative rules and meta-rules form the backbone of the KBS. The KBS further relies on the PE for performance data of the system. It diagnoses the bottlenecks in system operation from this data and suggests a tuning plan for performance improvement to the user. The KBS and the user interact through the user interface. The integration of the three phases is illustrated in the diagram below.

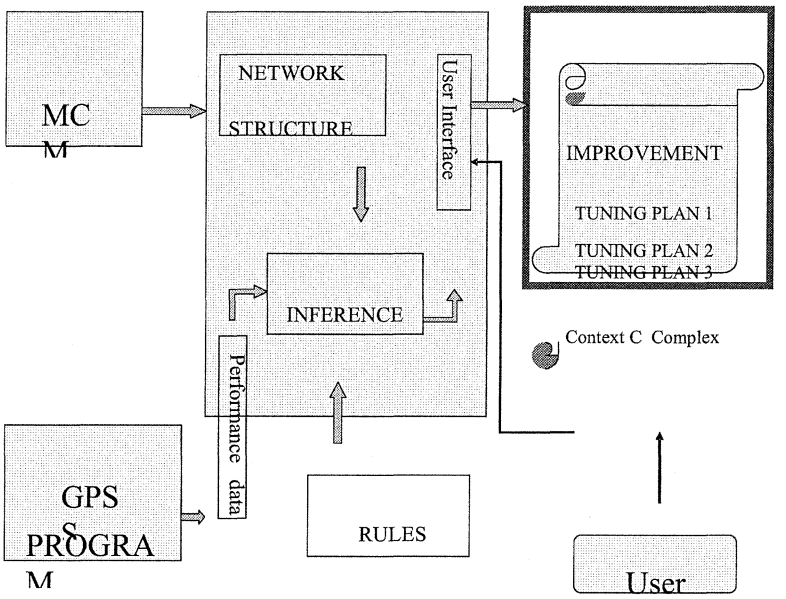

Figure 4. Conceptual Architecture for the Core Life Cycle

5. IMPLEMENTATION OF THE CORE LIFE CYCLE

We have attempted to create an integrated environment and to implement the three phases of modelling, performance evaluation and performance improvement (Gonsalves, T, Itoh, K, *et al.*, 2003, 2004a, 2004b). Microsoft Visio is used to create the descriptive model of the collaborative system. Arena, Simula, Simscript, GPSS, etc., are some of the well-known discrete-event simulation languages. We have

chosen GPSS (General Purpose Simulation System) for developing the system performance evaluating program, because of its versatility and ease-of-use. Finally, the bottleneck diagnosis and parameter-tuning plan are by the expert system constructed by using Prolog-based Flex toolkit. Each of the three steps in system analysis and design and the use of the respective tools are discussed in the following sections.

5.1 Modelling

Our model is a descriptive model of the collaborative system. The basic unit of collaborative activity is represented by a 'Context'. The actors in the system are known as 'Perspectives'. The requestor of activity is the 'Left-hand Perspective' and the performer of the activity is the 'Right-hand Perspective. There exists an interface between the two Perspectives through which Token, Material and Information (TMI) pass. These three represent three types of (related) tasks in the system. The topology of inter-connected Contexts gives rise to the 'Multi-Context Map' (MCM) model of the collaborative system (Hasegawa, A, Kumagai, S, *et al.*, 2000).

There are two semi-automatic software converters included as macros in the MCM drafter – the Prolog converter and the GPSS converter. The former converts the MCM network visual information into Prolog facts, while the latter converts the TMI flow and MCM contexts information into GPSS skeleton program. The GPSS converter is semi-automatic in the sense that the user has to supply the given parameters required to simulate the performance of the system.

5.2 Performance evaluation

Performance evaluation of the system operation is carried out by GPSS simulation. GPSS is built around abstract objects known as entities. A GPSS simulation program is a collection of a number of entities. The most prominent entity types are transactions and blocks. GPSS simulations consist of transactions and a series of blocks. Transactions move from block to block in a simulation in a manner which represents the real-world system that the designer is modelling.

The initial values of parameters such as service times, average inter-arrival time, the capacity of each server, the respective distributions for inter-arrival times and service times and the sequence of random number generators are provided. The system operation is simulated for the desired length of time and the required steady-state performance measures are recorded. The simulation is performed a large number of times to reduce the random fluctuations. The final simulation data sheet gives the performance measures averaged over a large number of simulation runs.

5.3 Performance improvement

Performance improvement is by the Flex expert system. Flex is a Prolog-based software system, specifically designed to facilitate the development and delivery of expert systems. A very useful and appealing feature of Flex is the use of Knowledge Specification Language (KSL) in programming.

The expert system consults the simulation data sheet obtained from GPSS simulation and diagnoses the bottleneck parameters. The bottlenecks that are detected are listed and displayed to the user. The user then selects desired number of bottlenecks for resolving. The inference engine of the expert system fires the appropriate knowledge-based qualitative rules to resolve the bottlenecks. It interacts with the user to obtain information regarding feasibility. It makes use of the

structural knowledge of the system obtained from MCM drafter to judge the propagation effects (advisability) of resolving the chosen bottlenecks. The output of the expert system is a set of performance parameters carefully selected for tuning.

6. CONCLUSION

Improving the performance of an established system is a cumbersome activity fraught with high risks and unjustifiable cost. Through the system development core life cycle we have indicated a way of projecting system performance estimate at the requirement analysis and design phase itself, much before the implementation phase of collaborative systems. The core life cycle consists of three phases: viz., system modelling, performance evaluation and performance improvement. The conceptual model of the system depicts the interrelationships between actors, tasks and collaborative activity in the system. Corresponding to these, actor sufficiency, task mobility and activity efficiency are chosen as performance measures of the system. Performance improvement is by a qualitative knowledge-based system. We have developed a systematic methodology for the conceptual core life cycle and indicated an implementation scheme for practical collaborative systems.

REFERENCES

Banks,J, Carson,II,J.S (1984) Discrete-Event System Simulation. New Jersey : Prentice-Hall.

Cooling,J (2003) Software Engineering for Real-time Systems. Addison-Wesley.

Gonsalves,T, Itoh,K,, Kawabata,R (2003) Performance Design and Improvement of Collaborative Engineering Systems by the application of Knowledge-Based Qualitative Reasoning, Knowledge Based Design Series, The ATLAS(1)

Gonsalves,T, Itoh,K,, Kawabata,R (2004) Perspective Allocating Qualitative Function for Performance Design and Improvement of Collaborative Engineering Systems (Proc ECEC2004, Hasselt, Belgium).

Gonsalves,T, Itoh,K,, Kawabata,R (2004) Use of Petri Nets in the Performance Design and Improvement of Collaborative Systems (Proc IDPT2004, Izmir).

Hasegawa,A, Kumagai,S,, Itoh,K (2000) Collaboration Task Analysis by Identifying Multi-Context and Collaborative Linkage. *CERA Journal* 8(1), pp61-71.

Itoh,K, Honiden,S, Sawamura,J,, Shida,K (1990) A Method for Diagnosis and Improvement on Bottleneck of Queuing Network by Qualitative and Quantitative Reasoning. *Journal of Artificial Intelligence* (Japanese) 5(1), pp92-105.

Kuipers,J (1994) Qualitative Reasoning Modeling and Simulation with Incomplete Knowledge. Cambridge : MIT Press.

Ross,S.M, (1990) A Course in Simulation. New York : Macmillan Publishing Company.