

Issues in Analyzing the Behavior of Event Dispatching Systems

Giovanni Bricconi
CEFRIEL
via Fucini 2
20133 Milano, Italy
bricconi@cefriel.it

Elisabetta Di Nitto
DEI, Politecnico di Milano
Piazza L. Da Vinci, 32
20133 Milano, Italy
dinitto@elet.polimi.it

Emma Tracanella
CEFRIEL
via Fucini 2
20133 Milano, Italy
tracanel@cefriel.it

Abstract

A good architecture is a necessary condition to guarantee that the expected levels of performance, availability, fault tolerance, and scalability are achieved by the implemented system. While in the software architecture domain several approaches focus on checking static characteristics of software systems, a general approach to performance analysis, independent from a specific application domain, is still missing. In this paper we focus on the analysis of performances of an event-based middleware with two objectives in mind: first, to determine the design alternative that provides the best performances; second, to experience with statistical analysis and simulation tools in order to understand their applicability to the evaluation of software architectures.

Keywords: Event notification systems, middleware, performance evaluation, software architectures.

1. Introduction

The architectural definition phase is a critical step of software development process, especially when the software being developed is composed of a set of distributed components. When the architecture is defined, several decisions are taken that can influence the way the implemented system addresses its functional and non-functional requirements. According to [2] a *good architecture is a necessary condition to guarantee that the expected levels of performance, availability, fault tolerance, scalability are achieved by the implemented system.*

We are currently experiencing the importance of a good architecture in the development of an event-based middleware called JEDI ([7] and [8]). JEDI supports communication among components (called *active objects*) through multicast of event notifications to all interested parties. The main components of the infrastructure is an

event dispatcher, that keeps track of all declaration of interests (*subscriptions*) issued by active objects and manages event propagation based on such declarations. The event dispatcher is implemented as a distributed system composed of several *dispatching servers* organized in a hierarchy.

Working on JEDI we have experienced that the way the dispatching servers are organized and collaborate – i.e., the way the architecture of the system is structured – has a strong impact on performance and scalability. For instance, intuitively, the hierarchical organization can improve performances compared to a centralized dispatcher. In some cases, however, it could happen that the traffic needed for coordinating the dispatching servers is higher than the traffic generated by the components connected to the middleware. This may result in undesired and unacceptable performance degradation. We have identified some solutions to this specific problem, but we have also realized that any solution needs to be carefully analyzed, in order to identify weaknesses and to find the best compromise. Indeed, it is desirable that this assessment is done before any of these choices has been actually implemented.

Software architecture research has focused on this a priori analysis and evaluation of design alternatives. In particular, [11] presents a methodology to guide the evaluation of architectural choices. This approach proposes an evaluation process that is mostly orthogonal to the specific evaluation techniques to be exploited. Other researchers propose some specific techniques. Most of them support checking of static properties, such as compatibility between communication mechanisms used by interacting components, or of dynamic properties, such as absence of deadlocks (see for instance [10] and [12]).

None of these approaches, however, is focused on evaluating architectures from the viewpoint of the properties of our interest, i.e., performances, network load, server utilization, response time, and scalability. A first proposal for the evaluation of these properties is presented in [14] where queuing networks are exploited to

determine metrics such as utilization of components and of connectors. Unfortunately, such approach does not fit well with our application domain, as we discuss in Section 3. Beside queuing networks, which are a well known analytical approach in the telecommunication and process scheduling fields, simulation is another important tool for performance evaluation, which is being extensively used in the design of some hardware and embedded systems and in telecommunication. Both simulation and analysis have been used in software development to deal with specific cases (see for instance [4], [1], [6], and [9]). What is still missing is a comprehensive set of guidelines and tools to support practitioners in the application of these techniques to the software architecture domain.

In this paper we present our experience in applying simulation and analytical techniques to our case study. Our contribution can be seen from different perspectives. From the perspective of developers of event-based middleware we propose proper tools for assessing and comparing different design approaches. From the software engineering perspective, we try to open a discussion in the community on the need for providing application-independent tools for performing quantitative analysis on software architectures.

The paper is structured as follows. Section 2 presents an overview of the event-based middleware we are developing and of the architectural alternatives we are facing with. Section 3 describes an analytical model for our system. Section 4 briefly describes the simulation model we have defined and the results that we have obtained from simulation. Finally, Section 5 provides some lessons we have learnt on the usage of simulation in the context of the architectural design of our system.

2. Event-based middleware and JEDI

Event-based middleware provide APIs and a run-time environment for development and operation of systems structured according to the *event-based style* [5]. In an event-based style, components communicate by generating and receiving *event notifications*. A component usually generates an event notification when it wants to let the “external world” know that some relevant event occurred in its internal state.¹ The relevant aspect in this process is that the event does not contain the list of its addressees: the event is propagated to any component that has declared interest in receiving it by issuing a

subscription, which is an expression on the content of the event. When an event respects the condition stated by a subscription we say that the event is *compatible* with that subscription. Every event issued by an AO is received by a connector, called *event dispatcher* or *bus*, which is in charge of identifying compatibles subscriptions and of notifying copies of the event to the AOs that issued these subscriptions. The event propagation is completely hidden to the component that generated the event. Thus, the event dispatcher implements a multicasting mechanism that fully decouples event generators from event receivers. This provides two important effects. First, a component can operate in the system without being aware of the existence of other components. All it has to know is the structure of the event notifications that are interesting to it, so that it can issue the necessary subscriptions. Second, it is always possible to plug a component in and out of the architecture without affecting the other components directly. These two effects guarantee a high compositionality and reconfigurability of a software building.

The middleware we have analyzed is JEDI (Java Event-based Distributed Infrastructure). Differently from some commercial tools, JEDI provides a quite expressive notification and subscription language. Event notifications have a name and a number of parameters of type string: for instance, `SoftwareReleased(Editor, 1.3, WinNT)` may be the notification describing that version 1.3 of a software called Editor for WindowsNT has been released. Subscriptions have the same structure of notifications, but their name and their parameters can contain the symbol “*” that represents a kind of wildcard. For instance, subscription `*(Editor, *, Win*)` matches with all the notifications (including the one above) having any name, three parameters, and concerning all the Editor versions that run on WindowsNT, Windows98, Windows2000...

As mentioned in the introduction, the JEDI event dispatcher is composed of a number of *dispatching servers* organized in a hierarchy. The distribution of dispatching servers is transparent to active objects (AOs for short). The system, in fact, guarantees that AOs receive all the notifications matching their subscriptions, regardless of the position in the dispatching hierarchy of the AOs that have issued them.

2.1 Subscriptions and event propagation

Whenever an AO issues a subscription, the dispatching server connected to that AO stores such subscription in its internal tables and forwards it to its parent server, which,

¹ In the following we will use the terms event notification and event indifferently, since from the viewpoint of event-based middleware we do not need to distinguish between the occurrence of an event and the generation of the corresponding notification.

in turn, stores the subscription in its internal tables and forwards it upward in the dispatching hierarchy till the root is reached.

When receiving a notification, a dispatching server forwards it to all its descendants (both dispatching servers and directly connected AOs) that have sent a subscription matching with it. In addition it sends the notification to its parent that, in turn, acts in a similar way (without sending the notification back to the original sub-tree). Therefore, a notification can reach each AO that has issued a compatible subscription, regardless of the AO position in the hierarchy.

The system guarantees that causally related notifications are received by AOs in the same order in which they have been generated. Moreover, it supports temporary disconnection and mobility of AOs. In particular when an AO disconnects, the system stores its subscriptions and incoming compatible notifications. When the AO reconnects, the system delivers to it all notifications that have been collected during its absence. For a detailed discussion of these issues see [8].

2.2 Improving the event propagation algorithm: advertisements

As discussed in the previous section, in the current version of JEDI events emitted by any AO reach the root of the dispatching hierarchy to guarantee that every subscriber in the system can receive them. This happens even if an event is interesting only for AOs connected to the same dispatching server (or subtree) of the original sender. So, when the number of events emitted in the time unit grows, the root dispatching server quickly becomes overloaded. To relief this situation we introduce a new event propagation algorithm that is based on the hypothesis that AOs perform a new operation called *advertisement*. An advertisement indicates the intent of an AO to generate specific sets of events. It is issued by the AO before it starts sending events of the corresponding type. Every AO can notify only events matching with the advertisements it has previously issued. Advertisements can be issued and withdrawn any time. The rules that define the matching between events and advertisements are the same holding between events and subscriptions (see previous Section 2). The introduction of advertisements allows dispatching servers to create proper routing tables, so that events are propagated only through paths leading to interested subscribers.

A critical issue is now how to propagate to all interested dispatching servers the information needed to create these routing tables. Carzaniga in [4] proposes some algorithms to deliver subscription and advertises in

networks of dispatching servers. These algorithms operate by flooding the network of dispatchers with copies of the subscriptions and advertisements issued by every AO. In this way, for each received event, a dispatcher knows if there are addressees and, if any, in which “directions” it has to forward copies of this event. To reduce the total number of advertisements and subscriptions to be distributed in the dispatching network, Carzaniga relies on partial ordering relations established between couples of advertisements and couples of subscriptions. Given a couple of advertisements (subscriptions) A and B, A *covers* B when all the events compatible with B are also compatible with A. Advertisements (subscriptions) covered by others advertisements (subscriptions) do not need to be propagated, thus reducing the overhead for handling them.

By taking advantage from the hierarchical organization of our dispatching network, we further optimize the algorithm presented above. We do not distribute subscriptions and advertisements to all dispatching servers, but only to a subset of them. More in detail, advertisements are routed toward the root of the dispatching hierarchy exactly as it was illustrated for subscriptions in the previous section, but the arrival of an advertisement in a dispatcher has some collateral effect. The dispatcher, in fact, forwards to the subtree that has generated the advertisement all subscriptions in its subscription table that are compatible with the incoming advertisement. We say that a subscription is *compatible* with an advertisement if and only if at least an event exists that is compatible with both of them.

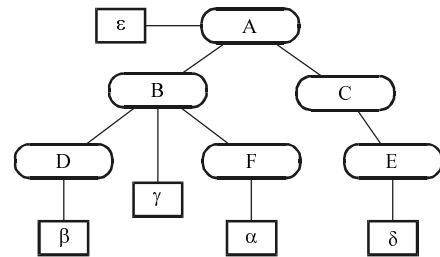


Figure 1. A dispatching server hierarchy.

As an example, let us consider the hierarchy shown in Figure 1. In the picture the boxes with rounded angles represent dispatching servers, while the rectangles are used to represent AOs. Let us suppose that AO β issues a subscription. According to the algorithm previously described, such subscription is notified to dispatchers D, B, and A. Now suppose that α issues an advertisement compatible with the subscription of β : This advertisement is received by F that forwards it to B, which, in turn, detects a matching with the subscription issued by β . Based on this, B propagates this subscription to F (that at this point is able to establish a route for all events

compatible with the issued advertisement), and, finally, propagates the advertisement to A. A, in turn, checks if other subscriptions, compatible with the advertisements, have been communicated by the subtree rooted at C. If not, it simply stores the advertisement in its internal tables. At this point, all events generated by α and compatible with the advertisement it has previously issued are routed to β through dispatcher B, and do not reach A unless A has sent a compatible subscription to B.

Of course, new subscriptions compatible with an advertisement can be issued even after an advertisement has been propagated (or during propagation). Therefore, the subscription distribution algorithm has to be modified too. Similarly to the case of advertisements, a dispatcher that receives a subscription, while routing it toward the root of the hierarchy, checks in its internal tables for compatible advertisements, and, if any, routes the subscription toward the senders of these advertisements.

3. An analytical model for JEDI

In order to understand the advantages provided by the introduction of advertisements, we have defined a statistical model that describes the behavior of JEDI. This model predicts the number of events that a dispatching server will receive per time unit, depending on the hierarchical structure it is inserted in and on the behavior of connected AOs. It gives an idea of the event traffic that has to be handled by each dispatching server given certain profiles of the input traffic of events and subscriptions. Intuitively, the best alternative to event propagation is the one that keeps the number of events received by the root of the hierarchy under a reasonable threshold in order to avoid the saturation of the system.

As in [14], we have evaluated the possibility of using queuing networks for modeling our system. Unfortunately, we have realized that this approach is not immediately applicable in our case. In fact, it is based on the hypothesis that there is no correlation between the types of messages that define the input traffic for the system. In our case, instead, such correlation exists. For instance, an unsubscription cannot precede a subscription and a notification has to reach an AO only if that AO has previously issued a compatible subscription. Another limiting assumption of queuing networks is that messages are assumed to be directed to one single destination, while of course in an event-based system they are often multicast to several destinations.

Nevertheless an analytic model, even a simplified one, is quite attractive because it would allow us to quickly obtain indications on the behavior of the system in terms of the load of its components.

Our model calculates the average number of events received by a specific dispatching server. In the model, to avoid the introduction of many compensative coefficients, we assume that every dispatcher controls the same number of AOs and that these are equivalent in term of probability of being subscribed to an event. In order to take the correlation between subscriptions and notifications into account, we introduce a coefficient called *spreading coefficient* (sc) that gives an indication of the distance to be covered by an event on the dispatching hierarchy. This coefficient is defined in the $[0, 1]$ interval and occurs in the following formula:

$$R(n) = sc^n \quad (1)$$

$R(n)$ is the probability, called *spreading probability*, that an event issued by an AO directly connected to a dispatcher A, has to be received (on the basis of the issued subscriptions) by some AO that is directly connected to some dispatcher at a distance n from A. The distance between dispatchers is calculated based on the topology of the dispatching hierarchy. For instance, in the topology of Figure 1 dispatchers A and D are at a distance 2. Intuitively, the more sc is close to 1, the more the probability that an event generated at a node connected to A reaches dispatcher D grows. In other words, when the value of sc is high, events tend to travel across the whole hierarchy, thus augmenting the load of the high level dispatching servers. When sc is low, events are mostly localized in some dispatching hierarchy sub-tree.

In order to calculate the average number of events received by each dispatcher, let us first focus on a pair of dispatching servers, D and S , at a distance $d(S, D)$ from each other.

The probability that D receives an event that has been generated by an AO connected to S is given by the following formula:

$$P(S, D) = P_1(S, D) + (1 - P_1(S, D)) \text{leaf}(D) P_2(S, D) \quad (2)$$

Where $P_1(S, D)$ is the probability that the event is delivered to an AO directly connected to D , $P_2(S, D)$ is the probability that the event is forwarded by D to some other connected dispatching server, and *leaf* is a function on dispatching servers that is defined as follows:

$$\text{leaf}(A) = \begin{cases} 0 & \text{if } A \text{ is a leaf of the tree} \\ 1 & \text{otherwise} \end{cases}$$

The introduction of *leaf* eliminates the second term of Formula 2 in case D is at the lowest level of the dispatching hierarchy. In this case, in fact, D receives only the events that are directed to the AOs it directly controls. Term $(1 - P_1(S, D))$ guarantees that events received by dispatching server D and forwarded both to other

connected servers and to some controlled AOs are not accounted twice in the formula.

If we assume each event reaches only one listener per level of distance, probability $P_1(S, D)$ is given by following formula:

$$P_1(S, D) = \frac{R(d(S, D))}{q_{S, d(S, D)}} \quad (3)$$

Where $R(d(S, D))$ is the spreading probability calculated at distance $d(S, D)$, and $q_{S, d(S, D)}$ is the number of dispatching servers placed at distance $d(S, D)$ from S .

To calculate $P_2(S, D)$ we have to consider the possibility that some AO, subscribed to the event under consideration, exists and is connected to dispatchers reachable from S only through D :

$$P_2(S, D) = 1 - \prod_{lev} \left(1 - \frac{k_{lev, S, D}}{q_{S, lev}} R(lev) \right) \quad (4)$$

In the above formula lev ranges from $d(S, D) + 1$ to the distance between S and the farthest dispatcher reachable through D . The term $k_{lev, S, D}$ represents the number of dispatchers at distance lev from S reachable only through D . Intuitively, Formula 4 defines the probability that an event has to be delivered to some dispatchers reachable through D in terms of the probability that this event is not required to reach any of the dispatchers reachable through D . Notice that Formula 4 (like Formula 3) holds only under the simplifying condition that the number of listeners for an event is limited to one per each level of distance.

We can exploit the result obtained through Formula 2 by defining a casual variable which assumes value 1 with probability $P(S, D)$ and 0 otherwise. Associating this casual variable to each event emitted by an AO, we can interpret its value as the arrival of the event in D . Based on the same principle, we can define a new casual variable X as the sum of all the casual variables corresponding to the events sent in the time unit from all the AOs directly connected to the dispatcher S . If all these events are independent from each other, X is a binomial casual variable. Exploiting the properties of the binomial distribution we can easily obtain the average number of events received by dispatcher D from the AOs connected to the dispatcher S . Let t be the number of events generated by the AOs connected to S in the time unit, we obtain:

$$E[X] = t P(S, D)$$

At this point, if we suppose that all the events are independent from each other, regardless their origin in the network, we can obtain the average number of events received by a dispatcher D in the time unit by simply

summing the contribution coming from every dispatching server in the network. Let us call this value $T(D)$:

$$T(D) = t \sum_S P(S, D) \quad (B1)$$

Where the sum is extended to every dispatcher in the network. Formula B1 holds under the following hypotheses:

1. Every dispatching server receives from its AOs the same number of events in the time unit.
2. The same spreading coefficient is adopted for every AO.
3. For each issued event, there is at most one AO subscribed per level of distance from the dispatching server at which the sending AO is connected.
4. Every event traverses a dispatching server only if it is necessary.

The last condition is true when advertisements are used, but it is false in the opposite case where events are always sent toward the root of the hierarchy. Fortunately it is easy to adapt the formula B1 to this case. If all the events are always sent toward the root, then every dispatcher D always (with probability equals to 1) receives a single copy of every event generated from the AOs connected to its descendants. Thus, we obtain the following formula:

$$T(D) = t \left(desc(D) + \sum_S P(S, D) \right) \quad (B2)$$

Where $desc(D)$ stands for the number of descendants of dispatching server D , and the sum is extended only to the remaining dispatchers of the network.

Condition expressed by item 3 is due to the fact that Formula 1 does not say how many addressees exist per level of distance from one dispatcher. Actually having more than one addressee for an event per level of distance leads to have two or more dispatching server committed in the delivery of the event thus increasing the probabilities defined by (3) and (4). The extension of the model to deal with these situations is still under development.

As an example of the results that can be obtained from the model, let us instantiate the model in the case in which the dispatching hierarchy is composed of 21 dispatchers organized in a quaternary balanced tree. Each dispatching server in this example is supposed to be connected to 10 AOs that generate events every 10 seconds on average (the intergeneration time has been modeled according to a Poisson distribution). The considered time window is of 390 seconds. Given this model, Figure 2 shows the number of events received by the root dispatching server plotted against the spreading coefficient. While the number of events is constant when the event propagation model is exclusively based on subscriptions (subscription-

based algorithm), it grows with the spreading coefficient when advertisements concur to the definition of the event routing tables (advertisement-based algorithm), but it stays below the threshold established by the subscription-based algorithm.

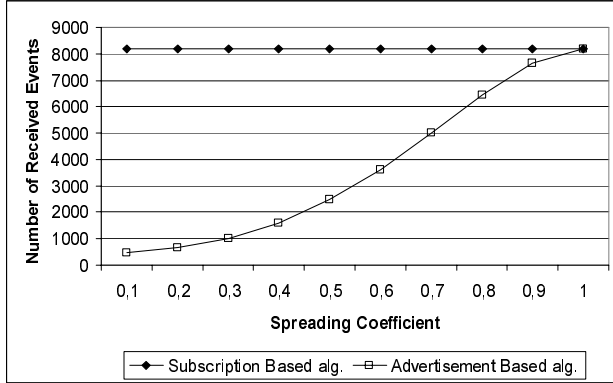


Figure 2. Events received by the root dispatching server.

From the results shown in the figure, the advertisement-based algorithm appears to be the most advantageous. Unfortunately the model does not provide complete information on the traffic due to the propagation of advertisements toward the root of the hierarchy and of subscriptions toward the advertising AOs. By taking these two traffic components into account, the advantage can decrease. As mentioned in the next section, however, the overhead introduced by this traffic is quite limited.

4. Simulating JEDI behavior

As we have discussed in the previous section, the usage of an analytical model to evaluate performance of our event-based infrastructure provides results that hold under simplifying conditions. A simulation approach allows us to: 1) validate the analytical model and understand its range of validity; 2) define a more complete model of the system in order to test different properties of it.

In general, the application of a simulation approach requires a simulation model to be defined. The simulation model is composed of a number of entities that define the characteristics of their counterpart in the “real world” that in our case is the JEDI infrastructure. The main entities of the model are dispatching servers and agents. Dispatching servers are characterized by a processing time associated to each of the messages they are able to receive (i.e., notifications, advertisements, subscriptions, ...). Agents model the behavior of AOs and are classified in four categories based on the type of traffic they generate. These categories have been derived by abstracting the

behaviors of AOs as they have been observed in some existing applications. They are:

- Data sources: they generate events at a rate that follows a Poisson distribution.
- Data sinks: they subscribe for some events and listen to receive them.
- Proactive agents: they issue some events and wait for some other event sent in reply.
- Reactive agents: they wait for an event and “answer” to it by issuing another event.

The environment we have selected to perform simulations is called OPNET [13]. In OPNET a model of the system to be simulated is defined by selecting components from proper libraries and by defining the way these components are connected together. OPNET provides a number of predefined libraries that model network components such as hubs, routers, and TCP/IP channels. Using these libraries it is possible to easily define models of local area networks as well as WANs and wireless systems with mobile objects and to simulate their behavior. In addition, OPNET allows the user to define his/her own libraries to model the behavior of specific application-dependent elements. We have exploited this feature to define our simulation model and we have relied on the existing libraries as for modeling the underlying network infrastructure.

Further details on the simulation model can be found in a companion paper [3].

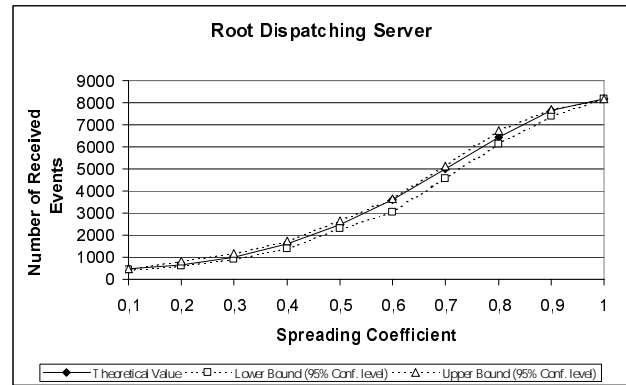


Figure 3. Comparison between the values obtained from the analytical and simulation models for the root dispatching server.

By running simulations on the above model, we have obtained results that show a good correspondence with the analytical data. For instance compare the theoretical and simulative results in the case in which the advertisement-based algorithm is used.

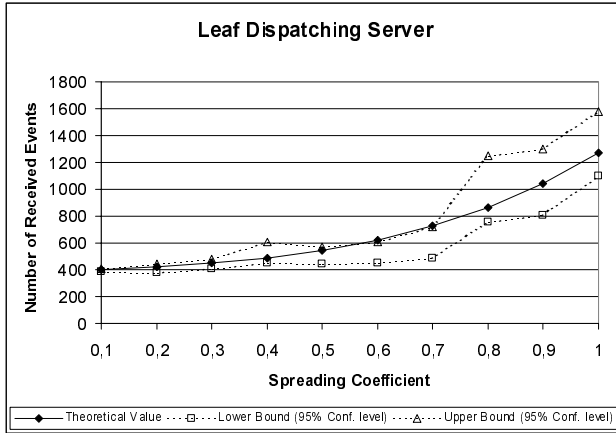


Figure 4. Comparison between the values obtained from the analytical and simulation models for a low level dispatching server.

The simulation has also shown that the overhead introduced by the advertisement-based algorithm is very limited. For instance, on the root dispatching server the additional load introduced by advertisements management results to be around 3% (see [3] for details).

Other results we have obtained through simulation concern the utilization of communication channels connecting the dispatching servers. Figure 5 shows these results when network channels of 64kbit/sec are used to connect 21 dispatching servers organized in a quaternary balanced tree. In this case, each dispatching server manages 52 agents located on 4 different hosts that transmit events with a frequency that depends on a Poisson distribution. The graphics focus on utilization of communication links connecting dispatching servers at different levels in the hierarchy. The graphics on the left-hand side refer to the traffic directed toward a dispatching server, while those on the right-hand side refer to the traffic directed in the opposite direction. Focusing on the graphics associated to the root dispatching servers, we can notice that the 64 kbit/sec channels get saturated for values of the spreading coefficient higher than 0.5 when the subscriptions-based approach is exploited. As for the traffic that exits the root dispatching server, we notice that it grows less when the subscriptions-based approach is adopted. This is due to the fact that, by exploiting advertisements, subscriptions need to be sent downward to establish the paths between senders and the subscribers.

5. Evaluation and conclusion

Event-based middleware are being designed to cope with the needs of highly distributed systems where components exchange a relevant number of asynchronous messages. Within this context, the evaluation of

performance and load level of both the network and the servers handling message distribution is very important since it provides a measure of the ability of the system to scale.

The exploitation of simulation and analytical models to evaluate such properties is certainly advantageous since it allows the system to be evaluated before it is actually build and deployed. The evaluation of the same properties on the deployed system would require much more resources. Moreover, if the evaluation indicated that the system provides poor performances, the system would have to be re-engineered. Conversely by exploiting simulation, we have been able to considered cases in which up to 4000 different interconnected computers where involved in the operation of the system.

Indeed, the combination of both analysis and simulation provides better chances to obtain results that are coherent with the system that is being evaluated since it permits such results to be cross-checked.

Nevertheless, using simulation and statistical models as tools to support system evaluation has some problems. Usually, when simulating telecommunication networks or hardware devices, traffic is assumed to be composed of messages that are independent from each other and are generated according to some well-known probability distribution. In our case, this assumption does not hold. In fact, traffic is composed of highly correlated messages. Such a correlation among messages cannot be disregarded and, unfortunately, it makes well-established analytical tools such as queuing networks hard to use. Most part of our effort in defining both the analytical and simulation model has been dedicated to solve this problem. As for simulation, we have built a piece of code that predefines (before starting the actual simulation) the correlation between all types of messages, based on parameters such as the spreading coefficient and others that account for possible similarities among different types of subscriptions (advertisements).

In other domains the exploitation of analytical and simulation tools is supported by a strong usage experience and by libraries and tools that hide the complexity of the underlying models to the users. As an example, we know from our colleagues that simulating some electronic circuits using VHDL is just a matter of composing existing pieces. Indeed, we have directly experienced that by using OPNET the definition and simulation of a TCP/IP network of any topology and speed is really easy to do and does not really require a deep knowledge of the internals of such kind of network. Unfortunately, we have also experienced a lack of similar support for simulating software systems built on top of these transport-level

protocols. In our case, we had to implement from scratch the behavior of any of our application-level components.

Similar easy to use libraries and tools should be defined for evaluating performance of software systems, possibly, independently of the application domain these systems belong to. These tools should be used either to compare predefined architectural alternatives or to play “what if” analysis that allow the designer to change the structure of the system until he/she finds a satisfactory performance level.

As for our on going work we are focusing on two directions. On the one side, we are consolidating the results obtained by simulating and analyzing the behavior of JEDI. On the other side, we are working on generalizing the analytical and simulation models we have built. The aim is to capture the characteristics of a variety of event-based systems. At the moment we are building tools that allow their users to select different dispatching server topologies and different event propagation models, and to perform “what if” analysis to identify the best topology and event propagation model, given the characteristics of the traffic to be managed by the dispatching network.

6. Acknowledgements

We wish to thank Prof. Alfonso Fuggetta for his valuable suggestions and ideas on the work we have presented in this paper, and Prof. Giuseppe Serazzi who gave us advice on the validity of our analytical and simulation models.

7. References

- [1] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R.E. Strom, and D.C. Sturman, “An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems”, *ICDCS '99* -- Int'l Conference on Distributed Computing Systems.
- [2] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, Addison-Wesley, the SEI Series, 1998.
- [3] G. Bricconi, E. Di Nitto, A. Fuggetta, and E. Tracanella, “Analysing the Behavior of Event Dispatching Systems through Simulation”, to appear on *HiPC 2000*.
- [4] A. Carzaniga, *Architectures for an Event Notification Service Scalable to Wide-area Networks*. PhD Thesis. Politecnico di Milano. December, 1998.
- [5] A. Carzaniga, E. Di Nitto, D.S. Rosenblum, and A.L. Wolf, “Issues in Supporting Event-Based Architectural Styles”, *ISAW3* -- 3rd International Software Architecture Workshop, Orlando, FL. November, 1998.
- [6] I. Chen and S.A. Banawan, “Performance and Stability Analysis of Multilevel Data Structures with Deferred Reorganization”, *IEEE Transactions on Software Engineering*, 25(5), September/October 1999.
- [7] G. Cugola, E. Di Nitto, and A. Fuggetta, “Exploiting an Event-Based Infrastructure to Develop Complex Distributed Systems”, *ICSE 98* -- 20th International Conference on Software Engineering, Kyoto (Japan), April 1998.
- [8] G. Cugola, E. Di Nitto, and A. Fuggetta, “The JEDI event-based infrastructure and its application to the development of the OPSS WFMS”, to appear on *IEEE Transactions on Software Engineering*.
- [9] S. Frolund and P. Garg, “Design-Time Simulation of a Large-Scale, Distributed Object System”, *ACM Transactions on Modeling and Computer Simulation*, 8(4), October 1998.
- [10] P. Inverardi and A. L. Wolf, “Formal Specification and Analysis of Software Architectures using the Chemical Abstract Machine Model,” *IEEE Transactions on Software Engineering*, 21(4):373--386, April 1995.
- [11] R. Kazman, M. Klein, M. Barbacci, H. Lipson, T. Longstaff, and S. J. Carriere, “The Architecture Tradeoff Analysis Method,” *ICECCS98* -- Fourth IEEE International Conference on Engineering of Complex Computer Systems, pp.68-78, Monterey, USA, August 1998.
- [12] J. Magee, J. Kramer and D. Giannakopoulou, “Behaviour Analysis of Software Architectures”, *WICSA1* -- First Working IFIP Conference on Software Architecture, San Antonio, Texas, February 1999.
- [13] MIL3 Inc, *OPNET Documentation Manual*.
- [14] B. Spitznagel and D. Garlan, “Architecture-Based Performance Analysis”, *the 1998 Conference on Software Engineering and Knowledge Engineering*, June 1998.

8. Appendix

LINK UTILIZATION

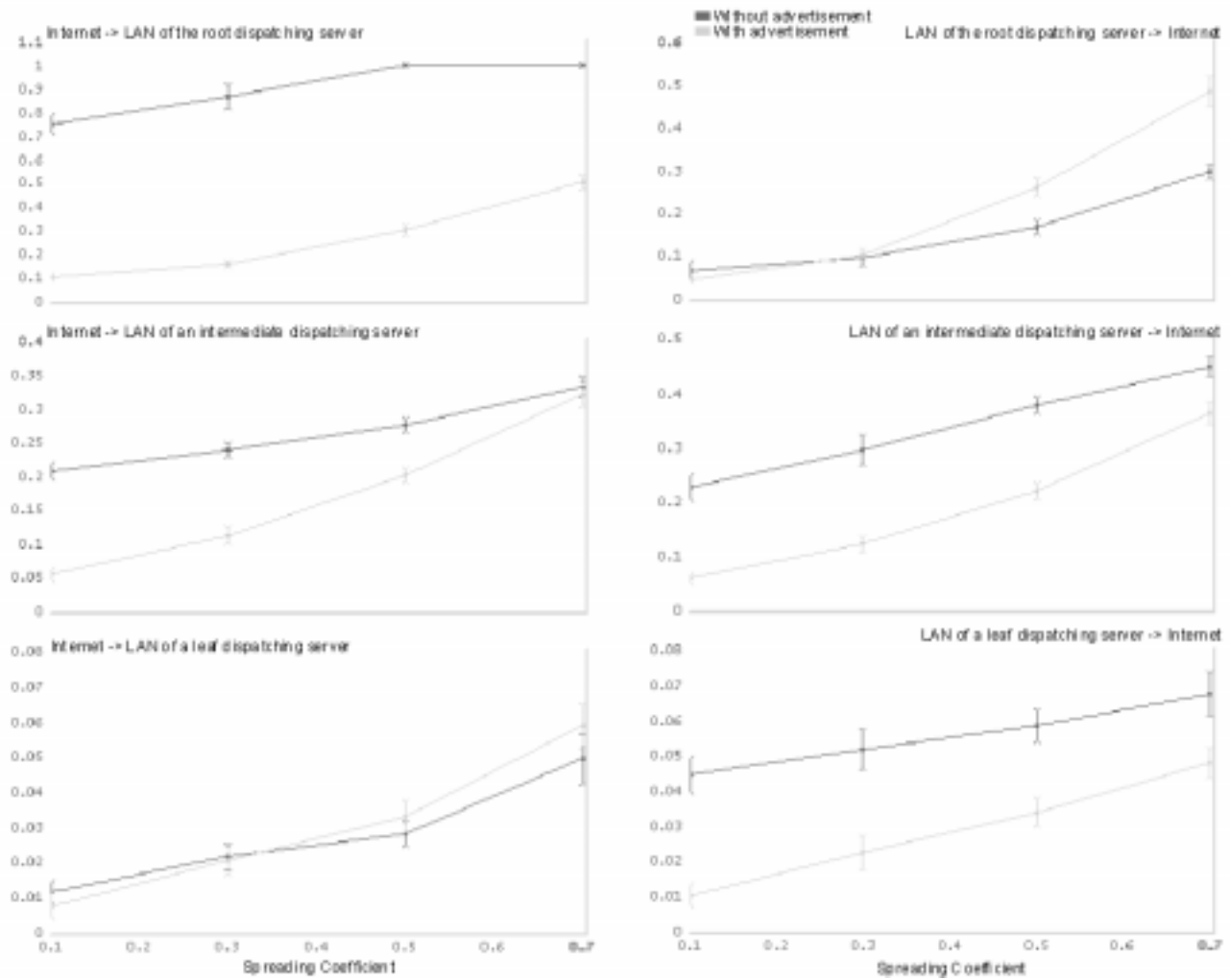


Figure 5. Utilization of communication links assuming communication channels of 64Kbit/sec.