Performance of a Publish/Subscribe Middleware for the Real-Time Distributed Control systems

Mohamed Anis MASTOURI and Salem HASNAOUI

SYSCOM Laboratory, National School of Engineering of Tunis TUNISIA

Summary

There's a world of opportunity for distributed embedded and real-time applications. The list of applications goes on and on: military systems, telecommunications, factory automation, traffic control, financial trading, medical imaging, building automation, consumer electronics, and more. These applications must find the right data, know where to send it, and deliver it to the right place at the right time. The publish-subscribe paradigm according to DDS is the best fit to such complex distributed applications that require a powerful communications model.

Thus, the goal idea of this paper is to study the defaults within a network of publish-subscribe nodes occurring in a clustered middleware, in order to calculate the loss rates while allowing for the caching size. A simulator has been developed in order to fix metrics chosen in the theoretical part.

Key words:

publish/subscribe, data-centric, Real-Time Middleware, Distributed Control Systems, caching, clustering, scalability

1. Introduction

Today's embedded software applications are increasingly distributed; they communicate data between many computing nodes in a networked system. This includes applications in aerospace, defence, industry, robotics, and telecom equipments.

These systems need new inter-object communication patterns. Several network middleware designs have arisen to meet the resulting communications need, including client-server, message passing, and publish-subscribe architectures. These later matches well with these patterns that need to send data from one producer to many consumers.

This paradigm is extremely attractive for structuring object-oriented Distributed Control Systems (DCS). It is a key enabler of the new distributed architecture. Data sources publish their data to the network; data users subscribe to the data to receive real-time updates.

The new Object Management Group (OMG) Data Distribution Service (DDS) standard is the first comprehensive specification available for "publish-subscribe" data-centric designs.

In this way, this paper tries to prove how Publish-subscribe mechanism is the best choice. It first provides an overview of DDS's functionality and compares it to other available technologies and standards. Then it predicts general design which show when DDS is the best networking solution when nodes are clustered. The caching problem has to be resolved to achieve the predictability for such applications.

The remainder of this paper is organized as follows: Section 2 outlines the advantages and the inconvenient of other middleware technologies that have emerged to meet the need of communication in DCS.

Section 3 illustrates the arguments in favour of the use of the publish-subscribe paradigm according to the DDS specification and we summarise several advantages of DDS's functionality that will be the focus of considerable research activities in the next few years;

Section 4 presents an overview of the architecture design of a middleware based on a publish-subscribe model and Section 5 gives the simulation results for the performance evaluation of a clustered publish-subscribe middleware using the caching techniques.

2. The Handicap of the Client-Server Model for such Architecture

In addition of the Publish/Subscribe paradigm, client-server paradigm is another model of communication which works well for object-centric applications.

2.1 The characteristics of Client-Server Communications

Client-server communications is characterized by a network resource (the server) that other network nodes (clients) access to get data or perform functions. Communications begin with a request by the client and end when the server replies. This is a popular model for enterprise applications that can rely on a rich set of processing and memory resources. It also fits well in applications where information or object services are naturally centralized.

Examples of client-server communications models include DCOM—Microsoft's Distributed Common and Object Model—and CORBA—the Object Management

Manuscript received January 5, 2007.
Manuscript revised January 25, 2007.

Group's (OMG) Common Object Request Broker Architecture. The OMG has updated the CORBA standard over the years to include real-time extensions and embedded versions of the specification.

The client-server model is best for applications where the service provider can be located anywhere on the network and, in fact, its location may change over time. The benefit is that the client and server code is separated so that each can be developed for different platforms and still be reusable. Client-server is an object-centric or service-centric view of a distributed system.

2.2 Drawbacks of Client-Server Communications

However, client-server has some drawbacks. Indeed, for the developers, while building their distributed real-time applications [2], the server represents a bottleneck and potential single point of failure. So, it cannot be fit for DCS. In addition, the request-reply semantics required two messages to get the data for each client. For those systems with many-to-many communication requirements, this resulted in high bandwidth load.

In addition, client-server architectures are often based around a remote method invocation or "object-centric" design. But in distributed real-time control applications, the information that needs to be communicated is quite often just data, not objects. Attempting to implement these "data-centric" systems with a client-server communications model frequently led to unnecessarily complex system designs and significantly degraded networking performance.

What was ultimately required was having a networking paradigm that introduced no bottlenecks, offered no single points of failure and lowered band-width loading for these mission-critical, data-centric applications. The publish-subscribe communication model fit these requirements. They can benefit from using both communication models as appropriate.

3. The benefit of using the Publish/Subscribe Paradigm

3.1 Characteristic and Arguments in favor of its use

Publish-subscribe is an ideal communication mechanism for moving data between distributed nodes of an application. In fact, a very large number of mission-critical, real-time control systems fit this model. [9] For example, distributed command and control systems with large amounts of periodic data are inherently quite data-centric. A sensor on the network periodically sends out data updates to controllers, loggers or other subscribers on the network. Publish-subscribe is almost a necessity for these systems.

Publish-subscribe is characterised by a set of data producers and data consumers. Whereas client-server has a request-reply form, publish-subscribe is more a push model. That is, after the publishers and subscribers have identified themselves on the network, the data is pushed by the publishers to the subscribers when new data is produced. There is no request; there is no polling. Like client-server, it has the similar advantage of making the system modular so code is highly reusable.

Another advantage is anonymous communications; publishers and subscribers don't need to know each other's physical network address. The middleware keeps track of which subscribers want which data from which publishers. Highly complex data distribution patterns are quite simple to actually program in this model. This anonymity also makes it simpler to set up redundant publishers for fault-tolerant systems. [8]

3.2 New Standard from OMG

The DDS is a new standard for distributed Real-Time Systems.[11] Its specification is governed by the Object Management Group (OMG) [14], which is the same organization that governs the specifications for CORBA, UML and many other standards. Since DDS is implemented as an infrastructure solution, it can be added as the communication interface for any software application. This DDS solution presents many advantages, in fact;

- It is based on a simple publish-subscribe communication paradigm
- It is flexible and adaptable architecture that supports "auto-discovery" of new or stale endpoint applications
- It produces a low overhead, so it can be used with high-performance systems
- It is characterized by a deterministic data delivery and a dynamically scalability.

DDS, and publish-subscribe in general, is already gaining acceptance, especially in mission-critical, distributed real-time systems in aerospace, defense and Telecom.

4. Design of the publish-subscribe model and the Appropriate middleware

Distributed real-time systems, especially distributed control and simulation applications are quite often data-centric. In the past, developers attempted to implement these data-centric applications using the CORBA client-server standard. This introduced inefficiencies and complexity. Thus, Publish-subscribe is a more natural model for representing and communicating data.

We have chosen to work within a platform of a Telecom equipment interconnection. This later will be the application of our conceived middleware based on publish-subscribe paradigm.

The Architecture of the application [3] is a set of nodes connected via a Real Time Transport protocol. In each node is embedded a Real Time Operating System, a middleware, and a Publish-Subscribe interface according to DDS specification. This is depicted in the following figure.

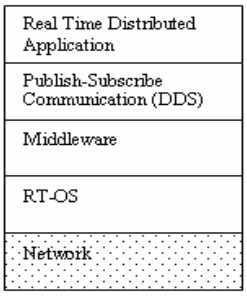

Fig. 1. An architecture of a distributed real time system using a publish-subscribe paradigm

The communication between nodes will be achieved due to publish subscribe interface via the Global Data Space that is represented by a relational data model. The middleware has to keep track of the data objects instances, which are considered as rows in a table. Each data object is identified by the combination of a topic and a topic-specified key. This is depicted in Figure 2.

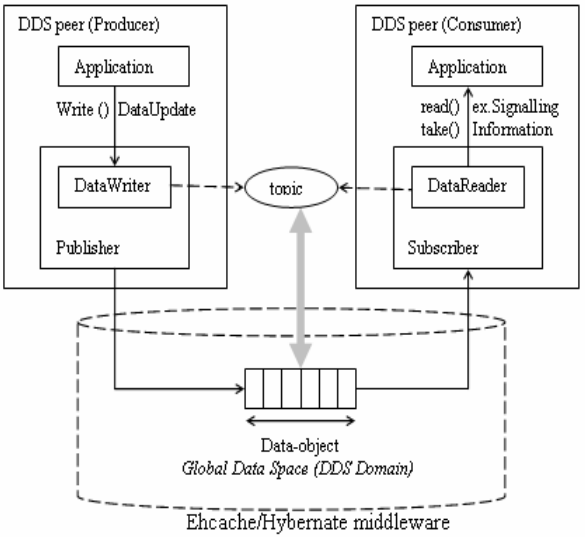

Fig. 2. Matching the topic with the adequate data-object via the middleware

This data-centric middleware allows the application to identify "data-objects" to the communication. The "data-objects" are unique in the 'global data space' of the distributed system across all participants.

Each participant is regarded as having a local cache of the global data-object. A message on a topic is regarded as an update to the data-object that can be identified and managed by the middleware.

Local changes to a data-object are propagated by the middleware; the middleware can distinguish between messages or update samples from different data-objects and manage their delivery to the interested participants on a per data-object basis. This scenario, in our implementation, will be achieved by a clustered distributed database based on ehcache and hibernate (Figure 3).

In fact the ehcache, in its last version (v1.2.1), will be used as a java library to speed up the processing time of the nodes.

Since, most of the processing time is getting data from a database. Therefore the speed up mostly depends on how much reuse a piece of data gets. In a system where each piece of data is used just once, it is zero. In a system where data is reused a lot, the speed up is large. This later is the case of our application, which will improve the performance of the processing thanks to the caching solution provided by ehcache and hibernate.

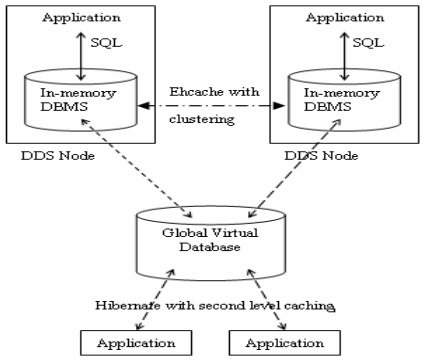

Fig. 3. Use of caching in the middleware design

5. Performance Evaluation of Defaults within Clustered Publish-Subscribe Nodes

5.1 The Application Platform

The publish/subscribe communication model is being the focus of many research in the last few years, notably with the increase need of the real time Distributed Control System – DCS in the Telecom applications.

Many researches have been performed to evaluate performances of distributed systems. Some of them are interested to study the publish/subscribe communication

model notably in the mobility context. [10] Other studies are interested to the data distribution in the real-time system, [17] but research of performance evaluation taking into account Distributed Control Systems including DDS and communication link failures for DCS systems have not been treated yet.

Our solution is based on the distributed caching among DDS nodes connected by a real time transport protocol (Figure 4). The caching will be useful to preserve the recent events for a given period of time.

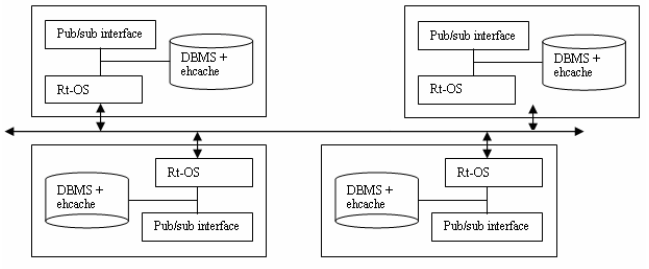

Fig. 4. The network of DDS nodes that will be evaluated

The analysis will be interested to the publish/subscribe paradigm according to DDS specification which is generally preferred for real-time DCS systems. Thus, the performance of this solution will be evaluated by:

- The analyses of the influence of errors in DDS nodes within the network simulated.
- Estimation of the average number of lost events per subscriber.
- Estimation of the cache size.
- Fixation of the failure level.

5.2 Theoretical analysis

In the performance analyses section we are fixed the following assumptions:

- When the link is disconnected during the publish/subscribe process, the data transfer is delayed until the link is reconnected, Thus publishers and subscribers wait until the recovery procedure is completed.
- When a subscriber fails, the lost events will be preserved in persistent logs managed by the Data Base Management System – DBMS. The subscriber can access events occurred during failure using theses persistent logs.
- When a publisher fails, another will replace it and the exchange of data won't be interrupted.

In order to measure the influence of errors to the clustered DDS architecture, we fixed the following metrics:

λ_{pub} : The publication rate,

λ_{sub} : The subscriber's access rate of published events.

λ_{fail}^{L} : The failure rate of the communication link

t_p and t_s : time delay for subscribe and publish. ($t = t_p + t_s$)

$\lambda_{\mathrm{Re}\,cov}^{sub}$ and $\lambda_{\mathrm{Re}\,cov}^{L}$: The recovery rate of the subscriber and the link

In the subscriber side, we suppose that i events are occurred during failure. When a DDS subscriber recovers from a failure it obtains its data from the DBMS which conserves persistent data. The probability that i events occurred between failure and recovery is:

$$p = \left(\frac{\lambda_{pub}}{\lambda_{pub} + \lambda_{re\,cov}^{sub}} \right)^{i} \frac{\lambda_{re\,cov}^{sub}}{\lambda_{pub} + \lambda_{re\,cov}^{sub}} \quad (1)$$

We note NL the maximum number of events that DBMS can store. If i> NL the events are lost. Thus, the average number of lost events by subscriber is:

$$E(N) = \sum_{i=N_L+1}^{\infty} \left(\frac{\lambda_{pub}}{\lambda_{pub} + \lambda_{re\,cov}^{sub}} \right)^{i} \frac{\lambda_{re\,cov}^{sub}}{\lambda_{pub} + \lambda_{re\,cov}^{sub}} (i - N_L)$$

$$\Rightarrow E(N) = \left(\frac{\lambda_{re\,cov}^{sub}}{\lambda_{pub} + \lambda_{re\,cov}^{sub}} \right)^{N_L} \frac{\lambda_{pub}}{\lambda_{re\,cov}^{sub}} \quad (2)$$

In case of subscriber failure, the system will be in an unknown state. Thus, on behalf of the subscriber, the loss of pursuit of the system's evolution is occurred during the following average time:

$$T = t_S E(N) = t_S \left(\frac{\lambda_{re\,cov}^{sub}}{\lambda_{pub} + \lambda_{re\,cov}^{sub}} \right)^{N_L} \frac{\lambda_{pub}}{\lambda_{re\,cov}^{sub}} \quad (3)$$

Likewise, in case of the failure link we could measure the cost of the model in a pulish/subscribe process on behalf of the subscriber. It is calculated by the addition of the publish subscribe delay (t_p+t_s) with the lost average delay which is multiplied by the probability to have a disconnection and the probability that a subscriber reaches events.

$$T = t + \left\{ \frac{\lambda_{Fail}^{L}}{\lambda_{recov}^{sub} + \lambda_{Fail}^{L}} \left(1 - \varepsilon^{-t\lambda_{Fail}^{L}} \right) + \frac{\lambda_{Fail}^{L}}{\lambda_{recov}^{sub} + \lambda_{Fail}^{L}} \right\} \left\{ \frac{\lambda_{sub}}{\lambda_{recov}^{sub} + \lambda_{sub}} \right\} \left\{ \frac{1}{\lambda_{recov}^{sub}} \right\} \quad (4)$$

Ultimately, to show the influence of the caching size and the rate of publication, we fixed the recovery rate of the subscriber to 0.5 and we measured the variation of the average number of the lost events according to the publication rate. We have increased at every time the size of the caching. This is depicted in the figure below. This later confirms the analysis conducted above.

The curve for N_L=0 represents a publish/subscribe system without durable database. This shows the

importance of the caching in such publish subscribe architecture.

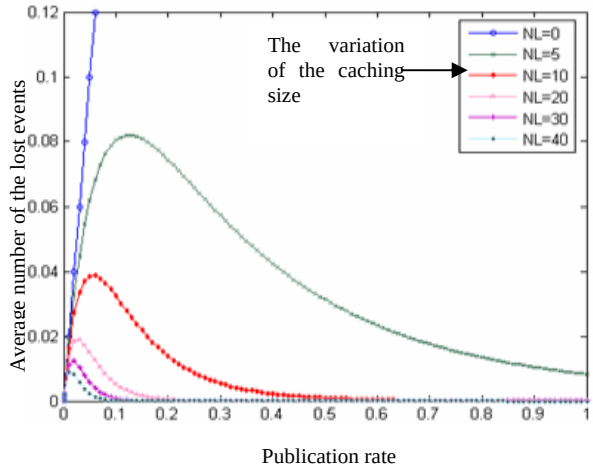

Fig. 5. The variation of the average number of lost events according to the caching size ($\lambda_{re\,\mathrm{cov}}^{sub} = 0.5$)

In the case of link failure, we have to fix the failure rate, to obtain the best performance for a fixed number of nodes.

In this way, the idea is to measure the ratio of failed connections according to the number of publishers and the frequency of establishing a new subscription. This permits to evaluate the unavailability of the system for the Subscriber in case of a link error with the publisher. This will be occurred in case of wired network.

Thus, the basic structure of the network comprises a set of publishers and subscribers containing a cache that will store a limited number of lost events.

In addition, we implemented a thread that is running continuously. Its task is to correct the failed links and to fail a random link between the subscriber and the publisher after a random period time. This is ensured with a class which extends TimerTask. This later is an abstract class that serves as the base class for the scheduled task. The failure process is implemented in the run method of the subclass. We also acted on the node class while assigning it a state of failure. This later represents the failure of the link between the publisher and the subscriber. The test of the link will be occurred every request for a new connection to the publisher. While the link is failed, the Subscriber repeats the request until the link is corrected. Moreover, the implementation will generate a file for each simulation time, in which we save the trace of the failed reconnections per subscriber according of the total number of connection. In addition we save the rate of failure for each connection number in another file. This rate will be evaluated according to the rate of the new subscriptions.

5.3 Simulation Results

In every iteration we vary two parameters, the new subscription rate and the number of publishers and subscribers, and we measure the failure rate.

- New subscription rate: it is the percentage of subscribers that establish new subscriptions (0.05 represents only 5% of subscribers that establish subscriptions).
- Number of publishers and subscribers: the number of publishers is chosen proportional to the number of subscribers.
- The Failure rate: this parameter represents the ratio of failed connections to the total number of connections. It represents an average for a fixed number of iterations.

The total number of connections depends on the two parameters "New subscription rate" and "Number of publishers and subscribers" (it is incremented for every establishment of a new connection: demands of connection are first piled in a task list and thereafter activated by the scheduler of the simulator).

The table 1 depicts the ratio of failure according to the number of publishers in the network.

$$\text{Failure rate} = \frac{\sum \text{of failures per iteration}}{\sum \text{of connections per iteration}}$$

The failure rate is calculated as follows (For new subscriptions =0, 05):

Table 1: The variation of the failure rate

Number of Publishers	Number of Subscribers	Number of Failures	Total Number of connections	Failure rate
4	40	21	49	0.42857
16	160	17	195	0.08717
36	360	5	400	0.0125
64	640	10	737	0.01356

The values obtained from the simulation tests permit to fix the ranges of the values used in the analytical section.

5.4 Interpretation of the results

The results deducted from the simulation are logically expected. They confirm the theoretical analysis conducted above. The graphics, depicted in Figure 5, show effectiveness of durable database which logs events for the failure of subscriber or link. In fact without logging, a subscriber loses events occurred during its failure. Thus, the loss of pursuit of system's evolution can be reduced by increasing the size of caching.

6. Conclusion

Publish/subscribe is a widespread communication paradigm for asynchronous messaging that naturally fits the decoupled nature of real-time distributed systems, allowing simple and effective development of distributed applications. Thus, following this theoretical and practical survey, we are convinced that the use of clustered middleware based on publish/subscribe infrastructures is necessary to build efficient real-time DCS notably for the Telecom equipment. This can be a stimulus for discussion, and possibly a starting point for providing an efficient architecture for the new generation of the network interconnection equipments [18] [19].

References

[1] Vida Kianzad and Shuvra S.Bhattacharyya. “Multiprocessor Clustering for Embedded System Implementation”. Technical Report UMIACS-TR-2001-52, Institute for Advanced Computer Studies, University of Maryland at College Park, June 2001.

[2] Paul Pop, PhD Thesis on “Analysis and Synthesis of Communication-Intensive Heterogeneous Real-Time Systems. Institute of Technology, Linköping, 2003.

[3] Mohamed Anis Mastouri, Salem Hasnaoui, “Design of Switch Architecture According to the MSF Framework Using the VSI Interface and the CAN-IOP Protocol”, ACIDCA-ICMI’2005 conference proceedings, p 13.

[4] Roberto Baldoni and Antonino Virgillito. “Distributed Event Routing in Publish/Subscribe Communication Systems”. A Survey, Technical Report 15-2005.

[5] Rajgopal Kannan, Sudipta Sarangi, S.S.Iyengar and Lydia Ray. “Sensor-Centric Quality of Routing in Sensor Networks”. Proceedings of IEEE INFOCOM 2003.

[6] Yanyong Zhang, Mark S.Squillante, Anand Sivasubramaniam, Ramendra K.Sahoo, “Performance Implications of Failures in Large-Scale Cluster Scheduling”. jsspp 2004.

[7] Antonio Carzaniga and Alexander L.Wolf. “A Benchmark Suite for Distributed Publish/Subscribe Systems”. University of Colorado, Department of Computer Science, Technical Report CU-CS-927-02 April 2002.

[8] Greg Eisenhauer, Karsten Schwan and Fabian E.Bustamante. “Publish–Subscribe for High-Performance Computing”. Published by the IEEE Computer Society, FEBRUARY 2006.

[9]Patrick EUGSTER, PhD Thesis on “Type-Based Publish Subscribe”, EPFL, 2001.

[10] Jinling Wang, Jiannong Cao, Jing Li. “Supporting Mobile Clients in Publish/Subscribe Systems”. Journal of Parallel and Distributed Computing, Manuscript Draft, 2005.

[11] OMG, "Data Distribution Service for Real-Time Systems Specification", March 2004,

[12] T.Kim, K.Kim, G.Jeon and S.Hong. Integration Subscription-based and Connection-oriented Communication into the Embedded CORBA for the CAN Bus. In IEEE Real-Time Technology and Application Symposium, Washington D.C., USA, May 2000.

[13] Salem Hasnaoui, Proposition d'un protocole CAN-IOP pour utilisation sur de nouvelles architectures de routeurs IPv6, de commutateurs ATM et de Réseaux Industriels, Brevet SN 000003148 INNORPI 2003.

[14] Object Management Group- Manufacturing Domain Task, Data Acquisition from Industrial Systems specification, OMG document dtc/01-09-03, November 2002.

[15] Mohamed Anis Mastouri, Salem Hasnaoui, “Design of Switch Architecture According to the MSF Framework Using the VSI Interface and the CAN-IOP Protocol”, ACIDCA-ICMI’2005 conference proceedings, p 13.

[16] Roberto Baldoni, Roberto Beraldi, Leonardo Querzoni and Antonino Virgillito. “Subscription-Driven Self-Organization in Content-Based Publish/Subscribe”. Proceedings of the International Conference on Autonomic Computing (ICAC), 2004.

[17] C.Riquier, N.Ricard and C.Rousset. “DES (Data Exchange System), a publish/subscribe architecture for robotics”. First National Workshop on Control Architectures of Robots, Montpellier, April 2006.

[18] Mohamed Anis Mastouri, Salem Hasnaoui, Abdelaziz Samet, a Virtual Switching Framework Using the CAN-IOP Protocol, CCCT05 conference proceedings, Vol.2, pp 268-274.

[19] Mohamed Anis Mastouri, Salem Hasnaoui, Abdelaziz Samet, Proposal of Master/slave Oriented Switching Control within the Context of Embedded ORB and the CAN-Bus, CCCT05 conference proceedings, Vol.2, pp 58-63.

Short Biography

Mohamed Anis MASTOURI was born in 1979. He received the Engineer diploma degree in computer science engineering from National School of Computer Science of Tunis (ENSI), in 2002, and the Master degree in communications systems from National School of Engineering of Tunis (ENIT), in 2004. Since this date he is preparing a PhD thesis on telecom in the Department of Computer Science and Communication Technologies of the National School of Engineering of Tunis (ENIT). Actually he works as assistant in the Faculty of sciences in Sfax, Tunisia. His research interest includes new generation of Telecom equipment, Data distribution service, publish/subscribe paradigm and real-time distributed control systems.

Dr. Salem Hasnaoui is a professor at the Department of Computer and Communication Technologies at the National School of Engineering of Tunis. He received the Engineer diploma degree in electrical and computer engineering from National School of Engineering of Tunis. He obtained a M.Sc. and third cycle doctorate in electrical engineering, in 1988 and 1993 respectively. The later is extended to a PhD. degree in telecommunications with a specialization in networks and real-time systems, in 2000. Prof. Salem Hasnaoui is author and co-author of more than 40 refereed publications, a patent and a book. His current research interests include real-time systems, sensor networks, QoS control & networking, adaptive distributed real-time middleware and protocols that provide performance-assured services in unpredictable environments. Prof. Salem Hasnaoui is the responsible of the research group "Networking and Distributed computing" within the Communications Systems Laboratory at the National School of Engineering of Tunis.

He served on many conference committees and journals reviewing processes and he is the designated inventor of the Patent "CAN Inter-Orb protocol- CIOP and a Transport Protocol for Data Distribution Service to be used over CAN, TTP , FlexCAN and FlexRay protocols".