社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2005－ARC－162 (1)
2005－HPC－101 (1)
2005／3／7

グリッド環境におけるモニタリングシステムの自律的構成

白勢 健一郎[†] 松岡 聡[†,††] 中田 秀基[†††,†]

グリッド環境での計算機資源のモニタリングは、効率の良い自動的な資源配分やグリッド環境の把握などに必要である。しかし、依存関係を持つ多数の構成要素から成り立っているため、その運用を人間の作業のみで行うのは限界がある。本研究では標準化団体の提案するモニタリングシステムのアーキテクチャに基づき、個々の構成要素が分散アルゴリズムなどを用いて自律的に設定やその更新を行うモデルを提案する。提案に基づいて既存のモニタリングシステムを対象にした管理機構を試作した。テストベッド環境の 14 個の PC クラスタのログインノードへの初期設定を 10 分程度で行い、障害修復の機能が動作することを確認した。

Autonomous Deployment of Grid Monitoring Systems

KEN'ICHIRO SHIROSE,[†] SATOSHI MATSUOKA[†,††]
and HIDEMOTO NAKADA[†††,†]

The problem with practical, large-scale deployment of Grid monitoring system is that, it takes considerable management cost and skills to maintain the level of quality required by production usage, since the monitoring system will be fundamentally be distributed, need to be running continuously, and will itself likely be affected by the various faults and dynamic reconfigurations of the Grid itself. Given our goal to develop a generalized autonomous management framework for Grid monitoring, we have built a prototype, on top of NWS, featuring automatic configuration of the components well as coping with single-node faults without user intervention. An experimental deployment on the Tokyo Institute of Technology's Campus Grid (The Titech Grid) consisting of over 15 sites and 800 processors has shown the system to be robust in handling faults and reconfigurations, automatically deriving an ideal configuration for the head login nodes of each PC cluster in about ten minutes.

1. はじめに

グリッドコンピューティングでは、物理的に分散した計算機資源を協調動作させ、実行するプログラムを並列化や適切な粒度に分割することで、大きな規模のプログラムの実行を目指している。そこで、全体の状況を基にした効果的なスケジューリングや、計算機資源の管理者やグリッド環境の利用者が、グリッド環境の現状を把握するために、各資源のモニタリングが必要である。

後で示す通り、グリッド環境のモニタリングシステムは、データの収集、集約、探索、解析などの機能毎に、構成要素を設けてこれらを協調動作させることによって実現する。したがって、モニタリングシステムの運用では、これらの構成要素間の依存関係を満たしつつ、グリッド環境の動的な変更に対応して、効果的な構成変更が必要となる。グリッド環境のモニタリングシステムに関しては、すでに技術標準化団体によって、一般的なアーキテクチャが示されている。したがって、これに基づいた一般的な運用の機構を考えることで多種のグリッド環境向けのモニタリングシステムに対応可能である。

IBM では Autonomic Computing[1] というコンセプトの下で、サーバの耐故障性、負荷分散の自律化の研究が行われているが、グリッド環境の管理を支援する研究は現在の所あまり知られていない。

これらの事を踏まえて、本論文では、集中管理モデルによってモニタリングシステムの運用を自動的に行う機構[2] に対し、より多くのモニタリングシステムに適用が容易で、標準化されたモニタリングシステムのアーキテクチャに基づいた、メタシステムによる分散管理の機構を提案する。提案に基づいて、既存のモニタリングシステムの実装をターゲットとした、管理機構のプロトタイプを実装し、動作検証や性能について検討した。

† 東京工業大学
Tokyo Institute of Technology
†† 国立情報学研究所
National Institute of Informatics
††† 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology

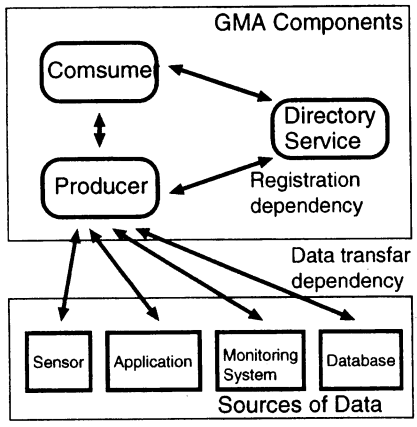

図 1 GMA の機能要素の関係

2. グリッド環境のモニタリングシステム

グリッドの技術標準化を行っている GGF の Grid Monitoring and Perfomance working group では "A Grid Monitoring Architecture(GMA)"[3] をまとめた。この文書では、グリッド環境のモニタリングシステムの基本的なアーキテクチャと構成要素間のインターフェースを規定している。

Producer 様々なセンサープログラムから性能計測データを受け取り、それらを他の構成要素に提供出来るようにする。GMA では Producer とセンタープログラム間のやりとりに付いては規定が無い。

Consumer Producer からデータを受け取り処理する。フィルタリングやアーカイバがこれに該当する。

Directory Service 他の構成要素の登録とその探索を支援する。

以上の構成は以下で述べるモニタリングシステムに当てはまるものであり、この構成に基づいて管理機構の設計と実装を行えば、多くのグリッド環境のモニタリングシステムに対応出来るものと考える。

この GMA に基づくとモニタリングシステムの構成要素の間には、以下の様な二つの依存関係のいずれかが存在する。

Directory Service と被登録要素の関係 Consumer のプログラムは、Directory Service への問い合わせによって、必要な情報を保持している Providor を探索する。したがって、Directory Service と Producer のいずれかについて、グリッド環境の変化に伴い、稼働場所や設定が変わった場合は、登録情報の更新や再登録が必要である。また、GMA では詳しく触れていない、Sources of Data に相当する構成要素も、Directory Service への登録が必要な場合が考えられるため、同様な対処が必要である。

データの送受信の依存関係 Producer – Consumer 間と Producer – Sources of Data 間には、データの送受信の依存関係が存在する。いずれの場合にも片方の構成要素の稼働場所や設定の変更に応じて、もう一方の設定を変更する必要が出て来る。

以下では、モニタリングシステムの実装に付いて述べ、それぞれが、このアーキテクチャに対応している事を確認する。

2.1 Network Weather Service

Network Weather Service (NWS)[4] は分散環境の CPU 等の計算機資源やネットワークの帯域等の利用率を測定し、将来の利用予測を行うシステムである。

NWS は主に以下の様な構成要素から成り立っている。

ネームサーバ 他の構成要素についての情報を保持。

メモリホスト データを不揮発性の記憶領域に保持し、要求に応じてデータを他の要素に提供。

センサーホスト CPU メモリ等の利用率を計測し、メモリホストに送信

利用予測プログラム センサーホストの取得したデータから、将来の利用予測を行う。

2.2 Globus MDS

Globus Aliance では Globus Toolkit の一部の機能として、Monitoring and Discovery Service (MDS)[5] を提供している。MDS はグリッド環境でのスケーラブルで信頼性のある情報システムを目指している。MDS は二つの構成要素を持っている。

GRIS (Grid Resource Information Service) グリッド環境の個々の資源の情報を収集し、蓄積する。

GIIS (Grid Index Information Service) 情報システムの階層化を実現。他の GIIS や GRIS を下位のノードとして登録し、情報の探索を支援する。

2.3 R-GMA

R-GMA[6] は EU DataGrid プロジェクトで開発されている、GMA の基づいたモニタリング・情報システムである。R-GMA は問い合わせに関係データベースを用いる。GMA の Directory Service、Producer、Consumer の各構成要素に対応した、Registry、Producer、Consumer を Java のサーブレットとして実装している。サーブレット間は Web サービスのインターフェースを介して通信する。

表 1 に GMA を取り上げたモニタリングシステムの構成要素の対応を示す。

表 1 GMA と各モニタリングシステムの対応

	NWS	MDS	R-GMA
Directory Service	Nameserver	GIIS	Registry Servlet
Producer	Memoryhost	GRIS	Producer Servret
Consumer	Commands	Commands	Consumer Servlet
Source of Data	Sensorhost	GRIS & Some sensors	Some sensors

3. メタシステムのよるモニタリングシステムの自律的管理機構

前述したグリッド環境のモニタリングシステムの一般的な構成に基づいて、自律的な運用の要件と実現方法を述べる。

まず、モニタリングシステムの運用において、以下の事柄の実現が必要である。

適切な設定の自律的な実現と維持 上述の依存関係を満たした構成要素間の初期設定と、運用開始後の動的な環境変化に対する変更の実現が必要である。

複数のモニタリングシステムのサポート 一般に、実際の資源モニタリングを行うプログラムと、情報の管理と提供を行うプログラムが別であることが多い。したがって、一つの運用の枠組で、複数のモニタリングシステムや情報システムのそれぞれを安定して動作させるだけでなく、それらを正しく組み合わせて動作するように、相互の設定を行う機能が必要である。

動的なグリッド環境の規模拡大への対応 大規模なグリッド環境の実現の手段として、既存の複数のグリッド環境の統合が考えられる。この際に、双方で動作しているモニタリングシステムの設定を、短い停止時間で実現できる必要がある。

他のグリッド環境の管理機構との連係 モニタリングシステムはグリッド環境を構築する要素の一つであるから、他のグリッドミドルウエアの運用ツールや PSE 等との連係が欠かせない。

以上の要件に基づき、本論文では、各構成要素が個々の設定を自律的に行うモデルを提案する。

自己反映計算[7] の概念を基にして、モニタリングシステムの各構成要素の管理をモデル化した、メタ要素に対応する要素の設定に関わる決定とその反映を行わせる。メタ要素は自律分散システム[8] に基づいた、マルチキャスト型の通信機構を用いて、情報交換を行う。

初期設定においては、メタ要素間で必要な情報を交換しながら、分散アルゴリズム等を用いて必要な構成要素を起動する。この際、先に述べた様な依存関係のある要素については、適切なものが稼働して、その通知を受け取るまで、起動するのを待っている。

一度、必要な構成要素が起動すると、Directory Service の様に多くの計算機で稼働させる必要の無いメタ要素は、障害発生に備えてバックアップの選出を繰り返す。メタ要素は稼働中のものが出すハートビートが

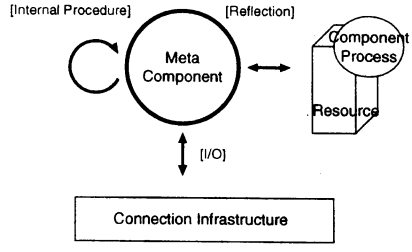

図 2 メタ要素の概念図

届かなくなった事や、明示的に稼働しなくなったという通知を受け取る事によって、バックアップを起動する。障害の修復に伴い、いくつかの構成要素の設定が変更されると、それらに依存関係を持っている他の構成要素は、依存先の構成要素の出すハートビートに含まれる情報から、設定を更新する。

グリッド環境の動的な統合や分離に関しては、管理者の異なる部分環境毎に管理機構を立ち上げ、管理機構間で相互に情報を交換する事で対応する。

以下で詳細に付いて述べる。

3.1 メタ要素

モニタリングシステムの各構成要素をモデル化した、メタ要素が対応する構成要素の設定に関する決定と反映を行う。メタ要素は次のものから構成される。概念図を図 2 に示す。

内部状態変数集合 メタ要素自身の状態、対応する構成要素のモデルや作用手続きに必要な情報、設定の決定を行うために用いるもの。

状態変更手続き メタ要素の状態遷移や分散アルゴリズムによる構成要素の設定決定の手続き。

作用手続き 構成要素の現状の把握、設定変更の反映など。

入出力 メタ要素間で情報の交換を行う時の通信手段。

メタ要素には構成要素の状態を運用する観点に基づいて、次の 5 つの状態が定義されており、状況の変化に応じて、この間を遷移する。5 つの状態の関係を記したものを、3 に示す。

初期状態 メタ要素が起動した際に初期化を行う。初期化が済むと設定状態に遷移する。

設定状態 他のメタ要素と情報交換を行い、必要に応じて分散アルゴリズムを用いて、対応する要素の設定に関する決定を行う。設定項目が一通り満たされた段階で構成要素を起動し、稼働状態に遷移する。Directory Service の様に全ての計算機で

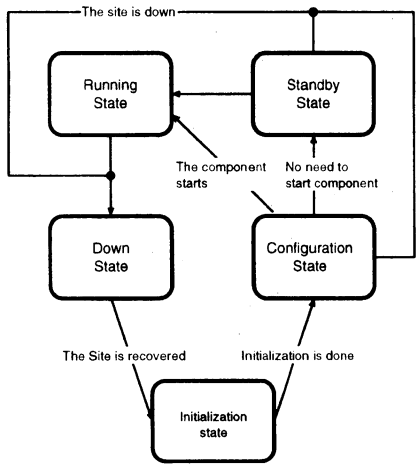

図 3 メタ要素の状態遷移図

動かす必要が無い構成要素の場合は、他の計算機で同じ種類のものが稼働しているかどうかを調べる。自らの所で動かす必要が無いと判断した上で、待機状態に遷移する場合がある。

待機状態 全ての計算機上で動かす必要の無い構成要素を対象とする。待機中のメタ要素の間でバックアップを選出し、障害が発生したときにすぐ対応できるようにする。バックアップが起動した時に稼働状態に遷移する。

稼働状態 依存関係のある他の構成要素に必要な情報を送信したり、逆に他の構成要素の稼働状態の変化に応じて設定を変更する。単に構成要素のプロセスが存在しない時には再起動させる。

停止状態 メタ要素が対応する要素と異なる計算機に存在する場合を対象にする。設定、待機、稼働のそれぞれの場合で、ネットワークが不通だったり、計算機が動いていない時にこの状態に遷移する。計算機が再び機能するのを待ち、復旧したのを検出した段階で初期状態に遷移する。必要に応じて、対応する構成要素が動いていないことを他に知らせる。メタ要素はこのような明示的な障害発生情報の他、一定時間を過ぎても稼働の通知が来なくなった場合に、障害が発生したと判断する機構を持つ。

3.2 メタ要素の動作と通信

メタ要素はイベントドリブン方式で動作する。具体的には情報を受け取ると、対応する要素の現在の状態を確認した上で、受け取った情報を処理する。次に処理した情報を基に設定変更の有無を検討し、変更の必要があれば、対応する要素に反映させる。最後に、他のメタ要素に通知する情報を送信して、新たな情報の到達を待つ。

メタ要素は自律分散システム[8] のアイデアに基づいて、ブロードキャストネットワークであるデータフィールドに接続している。データフィールドによって、一つの管理単位を構成する。この管理単位のそれぞれにデータフィールドゲートウエイを設け、複数の管理単位の動的な結合と分離を実現する。管理単位毎にポリシーを定めて不要な情報が管理単位をまたがらない様に出来る。これによって、1. 複数の資源 (PC クラスタ等) を集中的に管理する場合、2. 単独の資源を他の組織と協調して使用する場合、3. モニタリングシステムの各構成要素にメタ要素の機能を組み込んでしまう場合、のいずれにも対応することが出来る。

3.3 GMA を基にしたメタ要素の詳細

GMA を基にして各構成要素の自律的な管理の詳細を仮定を設けた上で述べる。

Directory Service まず、Directory Service はグリッド環境全体で一つあるものとする。それぞれの状態で、以下の様な手順を実行する。

- **初期状態** 評価値をそれぞれが求めた上で、互いに送信する。
- **設定状態** 後で述べるリーダ選出アルゴリズムなどを用いて、どの計算機で稼働させるかを決定する。
- **待機状態** 待機状態になったものの中で、設定状態の時と同じ方法で、最善のバックアップを選出し障害に備える。
- **稼働状態** 自分が稼働状態にあることを他に通知する。

Producer Producer はグリッド環境の資源をいくつかのグループに分割した上で、グループ毎に一つ稼働させることにする。グループ内での稼働場所の決定に付いてはリーダ選出アルゴリズムを用いる。以下に、各状態での処理を示す。

- **初期状態** 評価値を求めて、互いに送信する。
- **設定状態** ネットワーク上の距離などを基にして、グループ分けを行い、実際に Producer を動かす計算機を決定する。
- **待機状態** グループの更新と、バックアップの選出を行い、稼働中の Producer の障害に備える。
- **稼働状態** Directory Service の稼働場所や設定に変更が無いかどうか確認し、必要に応じて設定を変更する。また、自分が稼働状態にあることを他に通知する。

Sources of Data, Consumer Sources of Data はモニタリングを行う全ての計算機で稼働させる。Consumer に関しては人間が必要に応じて任意の場所で起動するものとする。設定状態でそれぞれ必要な他の構成要素の設定情報が届くのを待つ。稼働状態では依存している要素について、稼働場所や設定の変更の有無を調べ、必要に応じて再起動する。

4. モニタリングシステムの自律的構成機構の実装

提案に基づき NWS を主な対象として、Java を用いた管理機構の試作を行った。

一つの管理機構で複数の計算機の NWS の設定を実現できる様に、メタ要素の入出力の部分とデータゲートウエイをスレッドとしている。メタ要素のスレッドは内部に FIFO のキューを持ち、キューが情報を受信すると、個別に実装されたメタ要素の処理ルーチンを呼び出す。処理ルーチンは終了すると、他のメタ要素に送信する情報を、このスレッドに返す。スレッドはそれを送信してから、新たに情報を受信するのを待つ。

それぞれのメタ要素の処理ルーチンは、5 つの状態のそれぞれで実行する処理を、実装することになる。新たに扱う構成要素が増えるときは、この部分の実装を行えば、容易に追加が出来る。これらの処理ルーチンの実装は先に提案したものに沿って作られている。

以下で、プロトタイプ実装固有の項目に付いて述べる。

リーダ選出アルゴリズム 具体的なアルゴリズムとしては、最大の値を保持している要素を選出するアルゴリズム[9] を採用した。大小を比べる評価値としては稼働時間を用いている。

グループ分割 モニタされたデータの集約を局所的に行うために RTT を用いて計算機の遠近を定め、RTT が小さいもの同士でグループを構成させる。

Sensorhost の依存関係 Sensorhost は起動時に Nameserver と Memoryhost をそれぞれ指定する必要がある。したがって、設定状態の時は両方について情報が得られると起動を開始し、稼働中はいずれかの稼働場所や設定が変わった場合に、設定を更新して再起動する。

データフィールドゲートウエイ間の通信は UDP を用いる。このスレッドは内部で他からの受信用のスレッドと、送信用のスレッドを立ち上げ、送受信をそれぞれ独立して行う。

5. 実装したプロトタイプの評価

実装したプロトタイプに付いて、構成要素の配置や障害復旧の機能が正しく動作するかを検証し、また一連の処理にどの程度の時間が掛かるかを計測した。

構成要素の配置に関しては東京工業大学のキャンパスグリッド (Titech Grid) を用い、障害復旧に関しては松岡研究室の PC クラスタ (GSIC-Presto) を用いた。以下にそれぞれのシステムの仕様を示す。

表 2 実験環境の仕様

	Titech Grid	GSIC-Presto
CPU	Pentium III 1.4GHz	Athlon 1200MHz
Memory	1GB	1GB
NIC	100Base-TX	1000Base-T
OS	Red Hat Linux 7.0	Debian GNU/Linux 3.0
JDK	1.4.1	1.4.2

5.1 初期設定時の構成要素の配置

図 4 は 9 台の PC クラスタのログインノード上に NWS の構成要素を、配置した際の結果である。地区毎にグループが構成され、各グループに付き一つの Memoryhost が、稼働しているのを確認出来た。図 5 はログインノードの台数を 3 から 14 台まで変化させた時の所要時間と、(図のそれぞれ A が 3 台、B が 6 台、C が 9 台、D が 14 台の場合を表す。) 各構成要素が起動した時間を示している。処理の時間が分のオーダーになるのはメタ要素の動作を調べた結果、Memoryhost や Sensorhost のメタ要素が SSH と ping を用いて、RTT を集めている所に大部分の時間が使われている事が判明した。これは全てのログインノードに対して、逐次実行で ping コマンドを呼び出している事によるものと考えられる。Memoryhost のメタ要素に関しては、IP アドレスや、DNS のドメイン等を用いて、あらかじめ遠近の検討を付けること等によって、また Sensorhost のメタ要素に関しては、Memoryhost が稼働している所に計測対象をしぼることによって、オーバヘッドの削減が可能であると思われる。

5.2 障害復旧の機能の検証

図 6 に GSIC-Presto の管理機構を bwc03 で起動し、bwc00,bwc01,bwc02 の 3 台に NWS の設定を行った後に、Nameserver と Memoryhost の稼働している bwc02 をシャットダウンした後の、管理機構の動作を示す。SSH のタイムアウトを用いて計算機が稼働していないことを判断しているため、bwc03 で動いているメタ要素が bwc02 が停止したことを検出するのに、最大で 2 分程掛かっている。また、待機状態に入ったメタ要素も設定状態の時と同様、全ての計算機に対して RTT を測定している事により、新たな Nameserver や Memoryhost の稼働の後に、Sensorhost が設定を更新するのに 2 分掛かっている。以上の様に、計算機の稼働停止の判定や、RTT 収集によって、全体の処理時間が長くなっているものの、障害復旧を行う機能が動作することを確認した。

6. まとめと今後の課題

本研究では、グリッドの技術標準化団体の提案する一般的な、グリッド環境のためのモニタリングシステムのアーキテクチャに基づき、モニタリングシステムを構成する個々の要素に対応するメタな要素が、情報を交換しながら設定やその更新を行う自律分散管理のモデルを提案した。提案に基づいて NWS の初期設定と計算機の障害に対する、障害復旧機能をもつプロトタイプを実装した。実際のテストベッド環境でプロト

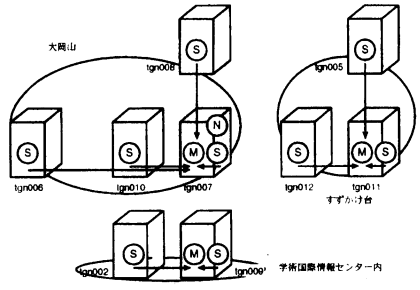

図 4 Titech Grid での NWS の構成要素配置の結果

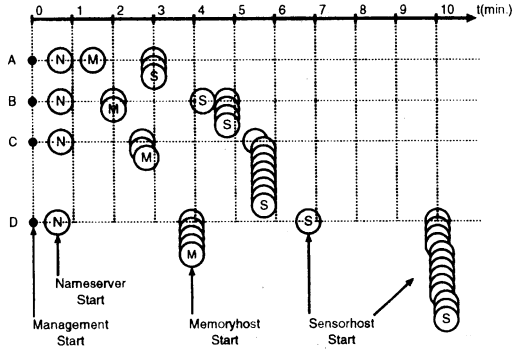

図 5 Titech Grid での NWS の初期設定の経過 A: ログインノード 3 台、B: 6 台、C: 9 台、D: 14 台

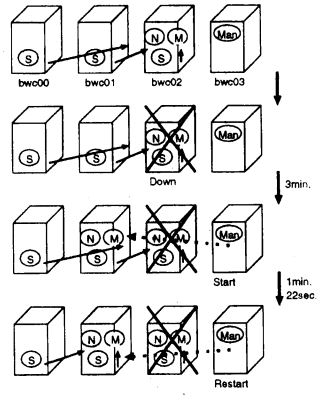

図 6 GSIC-Presto での障害復旧機能の動作検証

タイプの動作検証を行い、妥当な動作をすることを確認した。設定の決定に必要な情報収集に必要以上の時間を掛けているため、一連の処理が終るのに掛かる時間が長くなっているが、情報収集の仕方を工夫することで、性能の改善が見込まれる事が分かった。

今後の課題としては、今回実装したプロトタイプの性能改善、NWS 以外のモニタリングシステムにも少ないコストで対応できることの実証、より規模の大きな環境での評価が挙げられる。

謝　辞

本研究の一部は文部科学省「経済活性化のための重点技術開発プロジェクト」の一環として実施されている超高速コンピュータ網形成プロジェクト (NAREGI : National Research Grid Initiative) による。

参 考 文 献

1) R. Murch. *Autonomic Computing*, Prentice Hall, 2004
2) 白勢健一郎、小川宏高、中田秀基、松岡聡. グリッドコンピューティングにおけるモニタリングシステムの自律的構成, 情報処理学会研究報告 HPC-95, 2003
3) B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor and R. Wolski. *A Grid Monitoring Architecture*, Global Grid Forum, 2002.
4) R. Wolski, N. Spring and J. Hayes. *The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing*, Future Generation Computer Systems Vol. 15, 1999
5) K. Czajkowski, S. Fitzgerald, I. Foster and C. Kesselman. *Grid Information Services for Distributed Resource Sharing*, The Tenth IEEE International Symposium on High-Performance Distributed Computing, 2001.
6) A. Cooke, W. Nutt, J. Magowan, P. Taylor, J. Leake, R. Byrom, L. Field, S. Hicks, M. Soni, A. Wilson, R. Cordenonsi, L. Cornwall, A. Djaoui, S. Fisher, N. Podhorszki, B. Coghlan, S. Kenny, D. O'Callaghan and J. Ryan. *Relational Grid Monitoring Architecture (R-GMA)* UK e-Science All-Hands meeting, 2003
7) B. C. Smith. *Reflection and Semantics in Lisp*, 11th ACM Symposium on Principles of Programming Languages, 1984
8) K. Mori, H. Ihara, Y. Suzuki, K. Kawano, M. Koizumi, M. Orimo, K. Nakai and H. Nakanishi. *Autonomous Decentralized Software Structure and Its Application*, IEEE Fall Joint Computer Conference, 1986
9) N. Lynch. *Distributed Algorithms*, Morgan Kaufmann Publishers Inc., 1996