Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science 180 (2007) 91–106
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

# Intelligent Reconfiguration of Dynamic Distributed Components

George R. Ribeiro-Justo [1],[2]

*Capegemini UK Ltd., UK*

Ahmed Saleh[3] and Tereska Karran[4]

*Cavendish School of Computer Science,*
*University of Westminster,*
*London, UK*

**Abstract**

Naming service and the reconfiguration management systems usually work in a client-server manner where both the selection of components and the reconfiguration are a result of a request. Developing intelligent systems that are capable of monitoring and learning about themselves, and thereby rapidly react to changes in their environment, has become essential to most systems. This paper proposes an extension to FRODICA (Framework for Distributed Configurable Applications), a framework that supports the development of non-functional oriented reconfiguration of distributed systems, using the Complex Organic Distributed Architecture (CODA). CODA applies cybernetic concepts such as self-organisation, self-regulation and viability to derive an intelligent architecture, which can react to failures in achieving its objectives and proactively search for successful patterns of behaviour. The result is a configuration management system, which can use the knowledge of itself to proactively and dynamically drive both the search and the reconfiguration of its components.

*Keywords:* Complex adaptive system, business intelligence, reconfigurable distributed system

## 1 Introduction

The problem of adaptability has been studied for some time and we have previously investigated several aspects of reconfigurable distributed systems that can support adaptability [12] [14]. The premise is that dynamic changes in the system software architecture at runtime can increase adaptability. Further study has also shown that

[1] The views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied of Capgemini UK Ltd.

[2] Email: george.justo@capgemini.com

[3] Email: saleh@wmin.ac.uk

[4] Email: karrant@wmin.ac.uk

doi:10.1016/j.entcs.2006.08.039

those changes can provide increased adaptability when they support monitoring and tuning of the system's non-functional requirements [11]. After all, the architecture incorporates and is home to all non-functional requirements. Satisfying its non-functional requirements is not only essential for a system's success but also for its degree of adaptability.

The FRODICA (Framework for Distributed Configurable Applications) is part of an environment for the development of non-functional oriented reconfigurable distributed systems. FRODICA and its environment have been presented in details previously in [15] [11] and the present work build from that experience and proposes a new management system, which is more intelligent' and reactive in supporting dynamic reconfiguration. In this direction we have investigated several approaches to supporting intelligent adaptation and reconfiguration. Active entropy is an architectural paradigm that mimics the continuous, online feedback that is the hallmark of living organisms. Introspection devotes computational resources to observing system behaviour, then adapts the system accordingly [7]. The principle is that introspection adds information to the system in order to improve future behaviour. The IBM autonomic computing follows this principle. Autonomic computing focuses on the ways computers can carry out necessary tasks automatically, similarly to the way the human autonomic nervous system regulates breathing and other functions [9]. The system applies AI techniques to learn about the host computing environment and how it changes. If the system is having problems, the self-healing technology can send instructions as to how to implement a solution.

The autonomic approach is similar to the way CODA (The Complex Organic Distributed Architecture), described in this paper, works [13] [4]. CODA applies the cybernetic concepts proposed by the Viable System Model (VSM) [1]. The foundations underpinning the VSM are that all living systems are composed of a series of autonomous sub-systems, each having self-organizing and self-regulatory characteristics. They have within them the capacity to adapt to changes in their environment and to deal with the complexity that is relevant to them [2].

The paper proposes a new reconfiguration management system for FRODICA based on the concepts of CODA. In the next section, we present an overview of FRODICA and its management system. Section 3 introduces the cybernetics concepts behind CODA adaptability and how a CODA model can be defined. The main part of the paper is described in Section 4, where the new FRODICA intelligent reconfiguration management system discussed in detail. Finally, Section 5 presents the conclusions of this paper and directions for future work.

## 2 FRODICA Framework

Unlike the functional requirements of a system which describes what it should provide, non-functional requirements defines how it should provide its services and how it should behave at run-time. Non-functional requirements (NFRs) are as important as their functional counterparts, they do not simply represent quality properties that affect the system if not satisfied, but also play a crucial role in selecting the appropriate components for building an application and choosing between its alternative configurations. Despite their importance, NFRs are usually neglected whilst developing distributed systems due to the complexity of measuring, controlling and representing them. The explicit treatment of NFRs at the different stages of development, as well as at runtime requires some investigation about: 1) which NFRs to support, 2) how they can be represented, 3) how they can be measured, and 4) how they can be monitored/controlled at run-time.

The FRODICA (Framework for Distributed Configurable Applications) is part of a development environment for the development of non-functional oriented development of reconfigurable distributed systems [15]. The environment also includes two languages: an interface definition language to describe the functional and non-functional properties of a component's service (NIDL) and an architecture description language to define the application configuration. Measuring non-functional properties can be difficult and can increase overhead. Because of the nature of the applications FRODICA supports, three NFRs are treated: performance (transaction response time), availability and reliability.

FRODICA is a typical multi-layered component-based framework that offers a set of management components as well as a number of application-oriented components to handle common application functionalities. Although it is not the aim of the framework to provide full support for multimedia applications, FRODICA was initially built to support the distributed multimedia domain, thus contains a rich combination of distributed multimedia related aspects. The framework was later enhanced to support distributed configurable systems in general and non-functional oriented distributed systems in particular.

The framework architecture (Fig. 1) is divided into four consecutive layers based on the functionality and complexity of its components. The lowermost layer is called the *communication-layer* and it comprises all necessary components for handling lowlevel communications. The second layer is called the *general-purpose layer*, which deals with low-level system operations. The third layer is called the *application-oriented layer*, and offers all the standard services required for supporting the development of distributed systems. Finally the topmost layer is called the *specific application layer*, where all application and management components necessary for running a particular application are plugged together to form a meaningful architecture that meets the developer's requirements.

Application components/connectors can implement the framework interfaces (previously created by NIDL) that enable local management components at their machines to control their lifecycle. Components and connectors have to register with their local management components, which can then interrupt and control their execution. Components NFRs are monitored by an event-handling system (part of the runtime management system) located at the general-purpose layer of the framework. Information about components and connectors, as well as local management components, is stored in the *GlobalManager* (defined in the next section), which acts as a global information repository that simplifies the location of components within the system and provides a solid foundation for traceability (e.g.

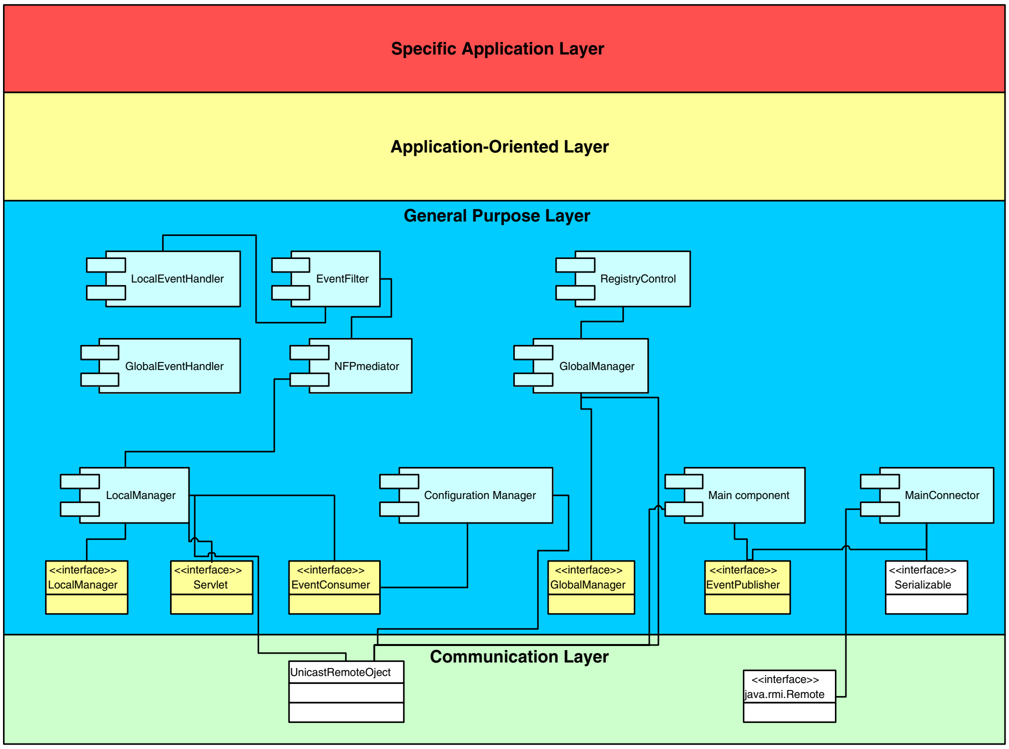


Fig. 1. The FRODICA overall architecture.

aids understanding, system maintenance and extension).

## *2.1 FRODICA Management System*

The management system is mainly located at the second layer of FRODICA. It comprises the *LocalManager*, the *GlobalManager*, the *ConfigurationManager*, the *RegistryControl* and the *NFRmediator*component (Fig. 1). The event-handling system, which coexists with the management system at the same layer comprises the *LocalEventHandler* and *GlobalEventHandler* component, an *EventFilter* component, and an *EventPublisher* and *EventConsumer*interface.

The key element for the monitoring and management of NFRs is the event handling system. Components usually generate events to indicate failure of NFRs or change of non-functional attribute states. In order to avoid excessive numbers of events being generated, events have to be filtered locally on each machine. The events that are not filtered, such as NFR failures, are forwarded to the (global) event system. Events are mainly consumed by the configuration manager, which interprets the NADL XML description and executes the necessary actions, for more details refer to [11] [16].

Events are mainly consumed by the *LocalManagers*, the *GlobalManager* and the *ConfigurationManager*. Events that have a direct impact on the system status are processed by the *ConfigurationManager*, which interprets the NADL-XML reconfiguration description upon the occurrence of an event and executes the necessary remedial actions through *LocalManagers*. After the successful completion of the reaction, *LocalManagers* have to report back to the *ConfigurationManager* to approve the reaction and update its status.

The *LocalManager* is one of the management components that have to be available on each machine where FRODICA components are running. Its main responsibilities are to:

- Load components on demand;
- Maintain a reference about components/connectors on its local machine;
- Detect any new/old component/connectors being added or removed at runtime;
- Collaborate with other *LocalManagers* to locate components with specific services/NFRs;
- Update the *GlobalManager* with newly available services/components;
- Carry out configuration/reconfiguration actions on behalf of the *ConfigurationManager*.

A copy of the *LocalManager* has to be kept on every machine within the FRODICA boundaries. It implements a registration system that enables each component joining/leaving FRODICA to register/unregister with its *LocalManager*. The *GlobalManager* is the core component of FRODICA that has to be instantiated before any of the framework components. It simply acts as a global information repository or trader that simplifies the process of locating components across the system. Its main responsibilities include:

- Maintaining a database of all components and *LocalManagers*;
- Loading *LocalManagers* on demand;
- Cooporating with *LocalManagers* in locating components with specific services/NFRs.

The information gathered by each *LocalManager* about its local components and their applications is passed to the *GlobalManager* together with a reference about the *LocalManager* and its location. The next section presents an overview of CODA, which is used to model the proposed reconfiguration management system.

## 3 Overview of CODA

A viable system is the one capable of independent existence [1]. To survive, a viable system needs not only a capacity to respond to familiar disturbances, but potentially to respond to unexpected, previously unknown disturbances. The latter is the hallmark of viable systems. It gives them the ability to adapt to changing environments [2]. Beer shows us how organisational structure  the necessary requirement for achieving purposes  can be given its essential property: viability.

A precise measure of (systemic) complexity has been defined as variety, meaning the number of distinguishable elements in a system, or by extension, the number of distinguishable systemic states. The state of a system at any time is the set of values held by its variables at that time. The *Law of Requisite Variety* established by Ashby [17] is used as a heuristic to develop criteria of effectiveness. Ashby's law describes the conditions under which a complex system can be externally controlled. The values of certain variables (essential variables) must remain within limits for the system to continue in existence.

Based on Ashby's law, Beer [1] defines the concept of the Viable System Model (VSM). The VSM is devised in terms of sets of interlocking Ashbean homeostats. In the VSM, the homeostats requisite variety is applied to the block of variety homeostatically related to the channels carrying information between them and to the transducers relaying information across boundaries. Every viable system can be seen as a collection of viable sub-systems together with a system that manages the relations between these viable lower-level activities so it is viable as a whole. Part of the variety is then pushed down to the next level and the remaining task is manageable. The proper tasks at each level are to formulate adaptive strategies and identify synergistic opportunities of the level immediately below. Key to the VSM theory is establishing that in any viable systems, there are five necessary and sufficient sub-systems involved in any organism or organisation [2], as illustrated in Fig. 2:

- **Implementation**: This subsystem encompasses the primary activities performed by an organisation, such as the production of products and services.
- **Co-ordination**: A viable system has a sub-system that co-ordinates the operations of its primary activities.
- **Control**: a viable system requires supervisory control supported mainly by a monitoring channel and the provision of an exception reporting system.
- **Intelligence**: This function focuses on the future, concerned with planning ahead.
- **Policy**: This system defines the direction, values and raison-d'être of the organisation. The policies are based on selective information provided by the Control and Intelligence systems.

These sub-systems are organised as layers in CODA taking into account architectural concept of enterprise systems. Details of the layers will be presented later. The theory behind CODA has already been demonstrated elsewhere [13] [4], therefore the focus of this paper is to present a model for adaptive reconfiguration management system.

### *3.1 Modeling Adaptable Intelligent Architectures with CODA*

At the heart of the CODA model is the concept of a *Role*, as shown in Fig. 3. A *Role* denotes an agent playing a specific role in the system. The reason for using the concept of a role rather than an agent is to emphasize the idea that a role has access to certain tasks. This is important to enable the system to control what can be done at critical times, for instance when resources are scarce, and consequently certain tasks should not be executed. This is key to the concept of adaptability.

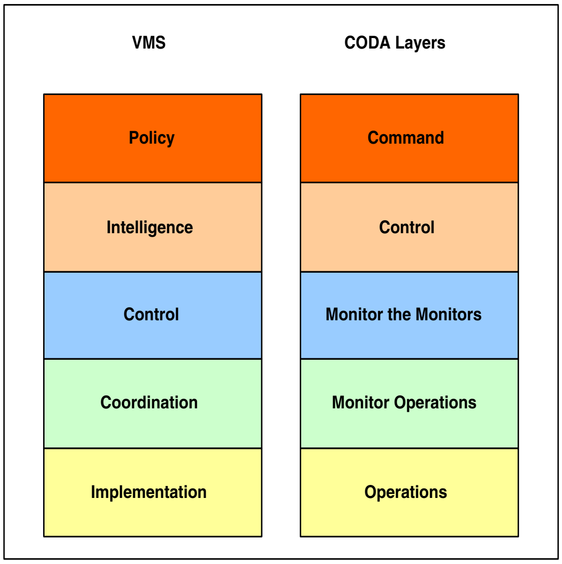


Fig. 2. Relationship between the VSM subsystems and the CODA layers.

A role can be seen as an active service, which performs tasks and operates on the goal of reaching critical successor factors (CSFs). A CSF is modelled as a measured objective. The CSFs define the SLAs (Service Level Agreements) of the service. For example, 90% of the services should satisfy their performance non-functional requirements, as shown in the example in Section 4.

The role's tasks are constrained by its filters, which determine the information that can be passed between layers (this concept will be discussed in more detail later) but more importantly establish the current CSFs the role should achieve. The filters are another mechanism used to support adaptability. For example, if the system is failing to provide the service that require high performance non-functional requirements, it may block access to other services to make more resources available. Short and long time memory are denoted by a *Wrapper*, which stores both current and historical events of the role. These events indicate the successes and failures of the role and are crucial to enable the system to adapt. This section has briefly introduced the key concepts of the model.

## *3.2 Modeling Autonomy*

The key principle behind the concept of role is autonomy. A role should perform its tasks autonomously, provided its CSFs are satisfied. If a CSF fails, the role will need assistance from other roles, possibly a higher-level role, which should have access to information and tasks not normally available to it. The higher-level role should be able to adjust the operational parameters (specified as CSFs and filters) of the failed role, to enable it to adapt to the conditions that are causing the failure. If the higher level role also fails to achieve its CSFs, it similarly alerts another higher layer role. As will be shown later, the chain of roles can extend to as many layers as there are in the system and only if it is not possible for the system to decide what

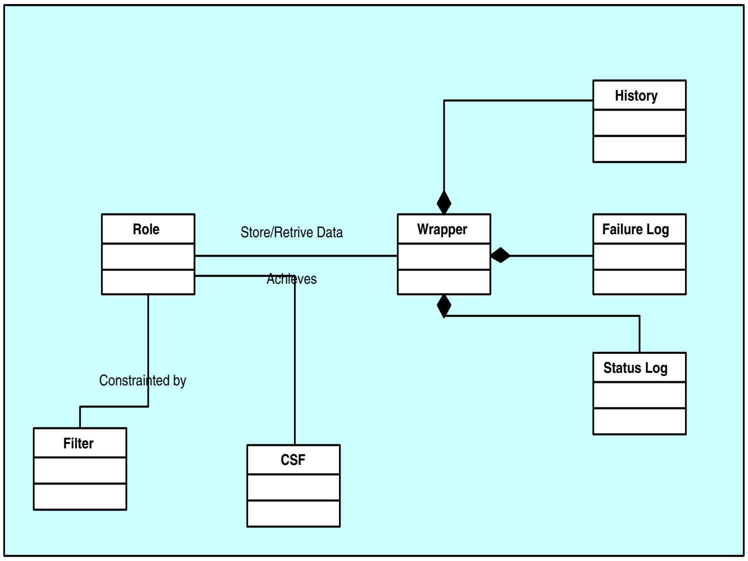


Fig. 3. CODA component model.

to do, will human intervention be needed.

### *3.3 Modeling Reactive Behaviour*

A role can be seen as a control component, which tries to achieve its objectives following the current operational parameters. If the role is unable to respond using those parameters, an alert is generated. The alert basically corresponds to a help call' to a more intelligent' component (higher layer role), which may assist the failed role in reacting to the event that has caused a CSF failure. CODA feedback loop mechanism allows the higher layer role to respond by resetting the failed role's operational parameters by adding or removing filters.

The concept of feedback loop in control systems is well known [1]. The CODA concept of feedback loop is more sophisticated because of its notion of systemic learning. Rather than just adjusting the operational parameters to cope with environment changes like traditional control systems, CODA uses its memory, the information provided by the wrappers, to identify the successful operational parameters. In this sense, the model presents a level of intelligence that is not encountered in typical control systems. In addition, because of the amount of information collated by system, the capability for learning is further enhanced even in comparison with other kinds of intelligent systems [5].

### *3.4 Modeling Proactive Behaviour*

In the previous section, the use of CODA concept of memory (current and historical data) was important to help the system to react, via the feedback loop, to unpredicted behaviour caused mainly by failure in CSFs. Historical data has another key function in the model, to support proactive behaviour.

An architecture modelled using CODA needs to make use of predictive techniques usually available in business intelligence [4]. This is essential for the system to predict its resource usage and successful services and thereby be proactive in adjusting CSFs and operational parameters. In addition, it is even possible to model emergent behaviour where the system can identify behaviour not previously known. For instance, the system may identify new user patterns or service usage.

## *3.5 Modeling Layers*

The concept of layers is fundamental in CODA. The principle is to separate the levels of intelligence of the system and also to categorise the data used by the system. CODA theory recommends a maximum five-layer model, following the VSM theory, as described earlier (see Fig. 2):

- **Operations**: This layer deals with simple linear data, which usually corresponds to the operational functions of the system. The operational data warehouse usually links together data from databases in several locations.
- **Monitor Operations**: In this layer, the data is often dimensional and aggregated. For instance, data is organised by time or group. This layer is responsible for monitoring business operations. Roles in this layer can react quickly to operational failures by filtering requests, which may affect CSFs.
- **Monitor (the) Monitors**: This layer deals with multidimensional data and provides capability for analysing trend behaviour. At this level, business operations are monitored in terms of external trends. It is possible to react more effectively by analysing the historical data from various levels of aggregations such as type, time and locations.
- **Control**: This layer should be able to learn about simple emergent behaviour, trends and forecasts and be able to run predictions and simulations automatically. Although the Monitor the Monitors layer can predict certain trends, this is actually modelled at this layer.
- **Command**: This is the highest layer, which should be able to deal with any variety not treated by the lower layers [17]. This means being able to recognise new threats and opportunities. Here we deal with strategic and long-term trends.

Although the above five layers are recommended for a complete adaptable enterprise architecture, it is possible to develop successful adaptable systems with only three of these layers, provided the Monitor the Monitors layer has predictive capabilities. It is important to observe that in fact most so-called adaptive' or reconfigurable systems only present two layers, which can be related to the two lowest layers of CODA [8]. However, these systems do not apply the concept of CSFs and filters, which are necessary for adaptability. Next section presents a new reconfiguration management for FRODICA based on CODA.

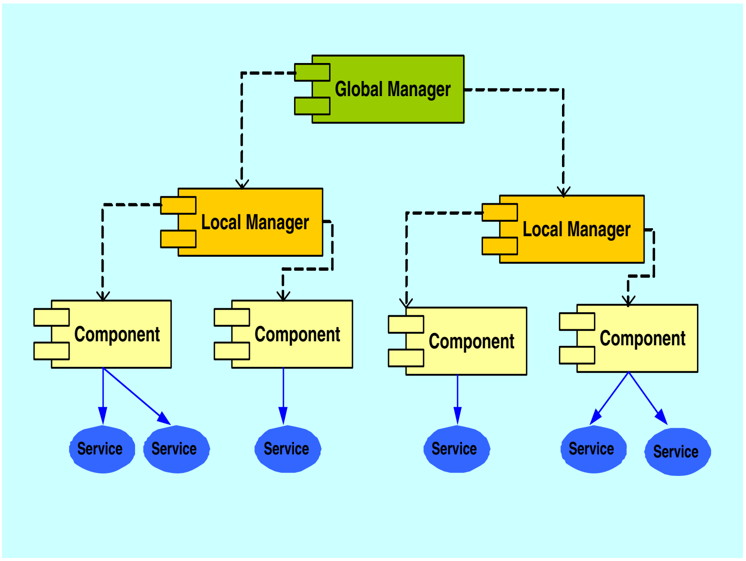


Fig. 4. The FRODICA management system hierarchical information structure.

## 4 FRODICA Intelligent Reconfiguration Management System

A key concept of a CODA model is the CSFs, which define the objective of the system. In the proposed FRODICA management system, the local manager can have autonomy as long as their CSFs are satisfied–that is, components are not failing their expected non-functional requirements. At the same time, when services are requested, the system is required to find services that can provide the expected non-functional requirements. Consequently, the system will be modelled in order to monitor itself and adjust its operational parameters to ensure that these objectives are met. The following section describes the details of the model.

### *4.1 Modeling Layers and Role*

We assume in this study that three CODA layers will be sufficient to provide the reactivity required by the FRODICA management system, as illustrated in Fig. 5. This assumption can be refined in later versions of the framework. Although CODA recommends a complete adaptable system to contain five layers, simpler systems do not require such a complete architecture, as explained in section 3.4. As mentioned earlier, each component encapsulates sufficient information about its services and NFRs, while each *LocalManager* maintains references to all components and connectors running on its local machine. Furthermore, the *GlobalManager* keeps records about each component and its associated applications and the *LocalManager* controlling its lifecycle. This hierarchical structure of information (Fig. 4) promotes extensibility to the framework and facilitates dynamic configuration, as new components/*LocalManagers* can join/leave the system at anytime.

The Operations layer is modelled using four roles (see Fig. 5), representing the

FRODICA *LocalManager*, *ConfigurationManager*, *GlobalManager* and event system (*GlobalEventHandler*) respectively. The naming convention we follow is to prefix the role name with the layer name. For instance the *LocalManager* role at the operations layers is named *OpLocalManager*. We do not explicitly model the concept of component type and service as roles because of their granularity. That would increase the size of the model.

The roles perform tasks and in doing so may request services from other roles. The system starts after the *OpConfigurationManager*loads an application configuration. The first service needed to build an application configuration is the lookup service provided by the *OpGlobalManager* role. It will try to locate a service with the required NFRs. The *OpLocalManager* contacts the *OpGlobalManager* on behalf of the *OpConfigurationManager*. The CSFs will determine if a service can be used and recommends where new services should be created.

If an existing service cannot be found, depending on the configuration, the *OpConfigurationManager* will request the *OpLocalManager* to initiate it by creating an instance of a component that can provide that service. The *OpLocalManager* can only create components depending on the constraints defined by its CSFs, as discussed later. After the component services are linked, the application starts. For simplicity, observe that the *OpConfigurationManager* requests are not filtered immediately but are instead controlled by the *OpLocalManager*. It could be possible to define application profiles with filters that could block requests from certain types of applications.

During the normal operation of the system, services are requested and their NFRs are monitored by the component. Service requests are recorded by the *OpLocalManager*in its Status log (see Fig. 5 and Fig. 6). If a particular NFR fails it generates an event that is propagated to the *OpConfigurationManager*, which performs the action defined by the application ADL script. The *OpLocalManager* is also informed and records it in its Failure log (see Fig. 5and Fig. 6). If the number of failures is greater than the CSFs threshold defined for the *OpLocalManager* it will generate an alert event that is caught by the monitoring layer *MLocalManager*. A new filter will be produced to that *OpLocalManager*, as we will see in the next section. If the new filter produced by the *MLocalManager* does not solve the problem, the *MLocalManager* will trigger an alert event to the monitoring the monitors M*MGlobalManager*.This role has a global view of the system, and can therefore make a better decision as to which *OpLocalManager* should be affected in order to achive the overall system's goals.

Observe in Fig. 5 that *MLocalManager* aggregates data from the *OpLocalManagers* and similarly the *MMGlobalManager* aggregates data from all *LocalManagers* and the *OpGlobalManager* as well. The data is usually transferred at defined time intervals. This will allow the system to recalculate roles's operational parameters (filters) even if alerts have not been generated. This allows emergent and proactive behaviour to be modelled. For instance, the system knows that at a specific time of the day there is a increased request for certain types of services, and so it can prepare itself ahead. In the next sections, we discuss the other elements of the model.

### 4.2 Modeling Critical Success Factors

CSFs are measurable goals. They are the parameters that enable the system to analyse itself and are key for reactive behaviour and autonomy. As long as the CSFs are satisfied the role will not depend on the intervention of others. The specific CSFs will depend on the system overall objectives. We assume that the main goal of FRODICA is to respond to all the requests for components that satisfy the given non-functional requirements. Because of the scarcity of resources, this may not be always possible and a certain level of failure is acceptable. Local managers are given a threshold, defined as a percentage of the number of services (components) that may fail. This is further refined into each non-functional requirement, that is; the number of services that fail their performance, reliability and availability. For instance, for the local manager Jaguar we expect 10% failure for performance, 20% for availability and 5% for reliability. It is important to stress that each local manager has its specific CSFs, depending on their resources. Fig. 6 illustrates the definition of the CSF within the filters.

Observe that the local managers only deal with running services but the system may also provide support for the global manager to locate or initiate new services. For instance, the system may satisfy its existing non-functional requirements but there is no spare capacity to accept any new requests.

CSFs are also categorised according to the dimensions in which the system will be self-analysed. For example, CSFs are specified for different time bands, when time is an important issue for the service provision. The system will therefore be able to analyse failures in terms of location (which local manager), time (including date) and type of non-functional requirement. Fig. 6 illustrates the definition of the CSF within the filters.

Survival CSFs are also defined for the system. They indicate if the system is reaching its capacity. In the case of FRODICA, this means the maximum number of (services) components of certain type running on the local managers. For simplicity, we assume that there is a correlation between the number of services and the amount of resources. This assumption will be revised in line with the work presented in [10].

### 4.3 Modeling the Feedback Loop

What happens when a CSF fails? As previously explained, the associated higher layer role needs to reset the failed role by virtue of its filters. When a local manager fails to offer services at the given non-functional requirement at the operations layer, it sends an alert to the local manager at the monitoring layer, which sends a new filter to the *OpLocalManager* to reduce the service offered. This basically means that it blocks all services that require the highest performance. If this does not solve the problem it blocks the services that require the highest availability and finally the ones that require the highest reliability. This also means that the global manager filter is also updated to prevent any new service request being allocated

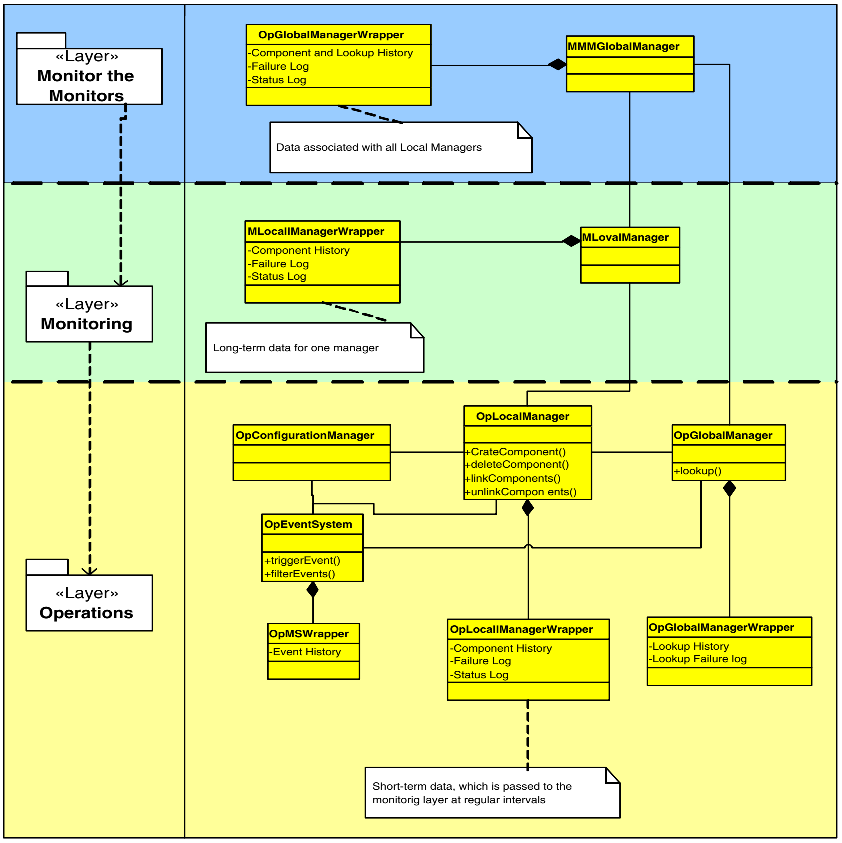


Fig. 5. The FRODICA intelligent management system.

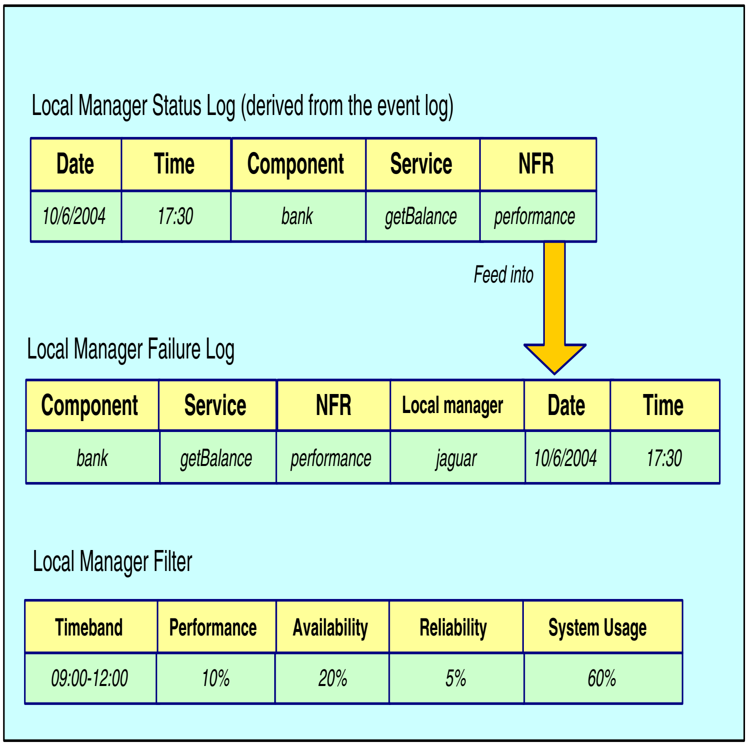
to that local manager. As explained in the next section, filters are dynamic and if failing local managers recover, their parameters are automatically adjusted.

### 4.4 Modeling the Filters

When a local manager first registers itself with the global manager it is given a filter, which indicates the number and types of services it should accept. The filter will depend on the type of local manager and the time of the registration. The filter is then further adjusted depending on the performance of the local manager, as discussed in the previous section. If their CSFs are failing, the filters will try to reduce the load on that local manager in order to try to improve its response to the expected quality of their services. Fig. 6 illustrates the definition of the CSF within the filters.

## 5 Related Work

Considerable research has been carried out in the area of dynamic adaptation and reconfiguration of systems but to our knowledge none of proposed solutions uses the idea of CSFs and multidimensional data. Narayanan [8] uses the concept of


Local Manager Status Log (derived from the event log)

| Date | Time | Component | Service | NFR |
|---|---|---|---|---|
| *10/6/2004* | *17:30* | *bank* | *getBalance* | *performance* |

Local Manager Failure Log

| Component | Service | NFR | Local manager | Date | Time |
|---|---|---|---|---|---|
| *bank* | *getBalance* | *performance* | *jaguar* | *10/6/2004* | *17:30* |

Local Manager Filter

| Timeband | Performance | Availability | Reliability | System Usage |
|---|---|---|---|---|
| *09:00-12:00* | *10%* | *20%* | *5%* | *60%* |

Fig. 6. FRODICA filters and wrappers.

fidelity' of the application in adapting the application resource consumption to the resource availability. This is generally achieved by using history-based predictions of the application behaviour and resource consumption. Historical data is used by data is not multidimensional and the system is very centralised. CODA uses distributed decision system using component CSFs and data warehousing techniques to manipulate multidimensional data. In additional, the operational environment is not reconfigurable as in CAST CODA.

Several researchers have investigated the use of reflective middleware in supporting adaptation [6]. The application can reconfigure the underlying middleware in order to enable a better use of resources or to provide a better QoS. In general, however, the research concentrates on developing mechanisms to support adaptation rather than the decision mechanisms and data used during the decision process. Our research is complementary, as a CODA can be integrated with a reconfigurable middleware to manage and co-ordinate the adaptation process. Recent research has been done in the area of architecture-based adaptation [3] where the system is monitored in terms of its architecture and then analysed to check if it behaves in an acceptable way. The research focuses on the definition of probes, to monitor the system, and gauges, to consume and interpret lower level probe measurements. Unfortunately, the authors state that the harder tasks, that of deciding on change and actually conducting the changes, are yet to be done'. The probes and gauges could interact with CODA in order to provide information and get advise in the same manner that the CAST reconfiguration controller can do.

In [10], the authors leverage their work on multi-fidelity and resource-aware application research to tackle the new problem of multi-component integration, configuration and reconfiguration, referred to as automatic reconfiguration. Unlike the traditional configuration task, which means building and installing new applications to the environment, in their approach configuration means to selecting and controlling applications so that the users' tasks are performed without disruption. In their terminology, the user's environment is made up of a set of devices and applications, which provide services and consume resources. In order to provide an acceptable level of services despite scarcity of resources, applications (often termed multi-fidelity or resource-aware) are designed to reduce the quality of operation and consume fewer resources. Their objectives are similar to our work but we model resources via their non-functional properties without referring directing to them. However, the most important difference from their work to the one presented in this paper is that our model uses not only current usage but also complex predictions based on multidimensional view of the past usage of resources.

## 6 Conclusions

The importance and need for reconfigurable distributed systems have been recognized for some time and a lot of progress has been achieved with both ADLs and their supporting reconfiguration system. The idea of supporting non-functional requirements or QoS is however more recent. FRODICA has introduced new ideas with its non-functional oriented IDL, ADL and reconfiguration management system. The idea of this paper was to introduce yet another novel feature which is missing in most reconfiguration management system, intelligence. The aim is to enable the reconfiguration management system to take more intelligent decision regarding the selection and provision of services.

The way we tackled this problem was by using CODA, a model for intelligent adaptive system we have developed within the context of EU-funded CAST project, which is aimed primarily at reconfigurable 4th generation mobile networks. However, CODA is a generic model and the paper has shown it can be applied to a wider variety of intelligent adaptive systems.

The use of historical data for modeling reconfiguration has been previously investigated but CODA uses more structured complex data and its layered architecture supports several levels of intelligence. In addition, CODA uses a set of data warehouses rather than simple files, which allows better analysis and forecast. In order to allow a better evaluation of the FRODICA management system intelligence, the data warehouses need to be fed with more operational data. This can also be done using simulation in order to generate more controlled data. It is also possible to refine the CODA model by introducing different CSFs and adding other types of non-functional requirements to the FRODICA.

Further investigation is also needed to add the higher CODA layer to the management system, which will support more powerful forecasting and level of intelligence. For instance, agents could be added to collect data on resources usage.

## References

[1] Beer, S., "Diagnosing The System for Organisations," Wiley, 1985.

[2] Espejo, R., and Gill, A., "The Viable System Model as a Framework for Understanding Organisations," Phrontis Limited & SYNCHO Limited, 1997.

[3] Garlan, S., Cheng, S., and Schmerl, B., Increasing System Dependability through Architecture-based Self-repair, in Architecting Dependable Systems, R. de Lemos, C. Gacek, A. Romanovsky (Eds), Springer-Verlag, 2003.

[4] Karran, T., Madani, K., and Ribeiro-Justo, G., Self-Learning and Adaptive Systems: The CODA Approach, in Software Defined Radio: Architectures, Systems and Functions, edited by M. Dillinger, K. Madani and N, Alonistioni, John Wiley & Sons Ltd,, 2003.

[5] Kayam, A., and Bailey, S., *Intelligent Architectures for Service-Oriented Solutions*, Web Services Journal, volume **03** issue 06, June 2003.

[6] Kon, F., Costa, F., Blair, G., and Campbell, R. H., The case for reflective middleware, Commun. ACM, volume **45**, pages 33–38, 2002.

[7] Kubiatowicz, J., *Extracting Guarantees from Chaos*, Communications of the ACM, Volume **46**, Number 2, February 2003.

[8] Narayanan, D., Flinn, J., and Satyanarayanan, M., Using History to Improve Mobile Application Adaptation, Proceedings of the Third Workshop on Mobile Computing Systems and Applications, Monterey, CA, December 2000.

[9] Paulson, L., *Computer System, Heal Thyself*, Computer, August 2002.

[10] Poladian, V., Souza, J., Garlan, D., and Shaw, M., *Dynamic Configuration of Resource-Aware Services* in Proceedings of the 26th International Conference on Software Engineering (ICSE), Edinburgh, Scotland, May 2004, pages 604–613.

[11] Ribeiro-Justo, G. R., and Saleh, A., Non-functional Integration and Coordination of Distributed Component Services, Six European Conference on Software maintenance and reengineering, March 11-13, 2002, Budapest, Hungary, IEEE Computer Society.

[12] Ribeiro-Justo, G. R., and Cunha, P. R. F., *An Architectural Framework for Evolving Distributed Systems*, Journal Of Systems Architecture **45**, 1375-1384, 1999.

[13] Ribeiro-Justo, G. R., and Karran, T., An Object-Oriented Organic Architecture for Next Generation Intelligent Reconfigurable Mobile Networks, 3rd International Symposium on Distributed Objects and Applications, DOA 2001, 17-20 September 2001, Rome, Italy, IEEE Computer Society.

[14] Rosa, N., Ribeiro-Justo, G. R., and Cunha, P. R.F., A Framework for building non-functional software architectures. In Proceedings of SAC 2001, ACM Press, Las Vegas, 2001.

[15] Saleh, A., Ribeiro-Justo, G. R. and Winter, S. C., *Non-functional Oriented Dynamic Integration Of Distributed Components*, in Proceedings of the 1st International workshop on foundations of Coordination Languages and Software Architecture (FOCLASA2002), Brno, Czech Republic, August 24th, 2002. Electronic Notes in Theoretical Computer Science Vol. **69** (2002).

[16] Saleh, A., "An Environment for Non-Functional Oriented Development of Distributed Configurable Architectures," PhD thesis, Cavendish School of Computer Science, University of Westminster, London, 2003.

[17] Waelchi, F., The VSM and Ashby's Law as Illuminations of Historical Management Thought, in The Viable System Model: Interpretations and Applications of Stafford Beer's VSM, R. Espejo and R. Harnden editors, Wiley, 1996.