Model-driven Management of Complex Systems

Brian Pickering[1], Sylvain Robert[2], Stéphane Ménoret[3], Erhan Mengusoglu[1]

[1] Pervasive & Advanced Messaging Technologies
IBM UK Laboratories, Hursley Park, Winchester, UK
[2] CEA LIST, Boîte courrier 65, Gif-sur-Yvette cedex, F-91191, Fr
[3] Thales Research and Technology, RD128, 91767 Palaiseau cedex, Fr
{brian_pickering, mengusog}@uk.ibm.com, sylvain.robert@cea.fr,
stephane.menoret@fr.thalesgroup.com

Abstract. This paper considers some specific issues relating to model-driven system management applied to complex systems. Examining dynamically coupled systems-of-systems on the one hand and highly distributed devices for service access on the other, we define a common meta-model of (semi-) automated management applicable in both domains. Taking monitoring by way of illustration, we then show how this meta-model is put into practice along two complementary aspects: management modelling and runtime event processing support.

Keywords: system management, runtime modelling, complex event processing.

1 Introduction

Previous work has looked at exploiting design-time architectural models at runtime in order to evaluate and validate potential changes to the current managed system [12], [8] and [9]. Although well motivated, because of power limitations [9] in the managed devices, or to check that potential changes would be of optimal use in the current environment [7] and so forth, there are issues about what can and cannot be captured at design-time. Kodase *et al* [10] suggest non-functional requirements are difficult to capture in design-time models, for instance; and Ulbrich *et al* [13] propose that quality-of-service management can only effectively be handled by message path manipulation during operation. In addition, most work from the seminal Oreizy *et al* paper on self-adaptation 11] to Rainbow [8] and the MADAM proposals for (mobile) telecommunication services [7] focus only on runtime modification of the managed system itself. There are, therefore, a number of important gaps here. Can we, for instance, introduce non-functional as well as functional aspects into our design time models? And can we now address omissions such as historical data, domain-specific rules and policy management (see the *Future Work* section in [7] for instance)? Most significantly, perhaps, can we make dynamic, context adaptive changes to the system management components using a design-time model just as we would for the managed system? This work attempts to address some of the issues raised in current model-driven approaches to system management. In this work, we seek to address some of these questions. To begin with, we introduce our autonomic approach to system and device management (Section 2) and present the top-level meta-model we are defining for system management. Next, we consider specific issues related to management and system runtime modelling (Section 3), and finally (Section 4) we consider how to allow for runtime adaptation of the management system itself and the introduction of dynamically changing functional as well as non-functional requirements.

2 Model-Based Management Common Framework

Within the context of the MODELPLEX project, we have begun to define and evaluate common models for system management [1]. Although sharing some features with other approaches, notably OASIS, SMDS and a number of OMG initiatives *op.cit.*, we have focused our work on common issues which affect different aspects of system management. One specific area of interest involves the automatic and semi-automatic management of complex systems.

For system management purposes, we recognize a number of key concepts as summarized in Fig 1. A *ManageableElement* is the central and fundamental object which may be defined as any and all elements within a system that need to be managed. Each element is associated with one or more *ManageabilityCapabilities* that describe what and how that element needs to and can be managed. The elements, though, are not confined to those concepts and objects which are subject to being monitored and controlled. We must also include elements of the management system themselves, as well as the definitions of the criteria and rules by which the system is managed. So we need to be aware that *ManageableElements* may well include *ManageableSystemElements* or *ManagementRules* respectively.

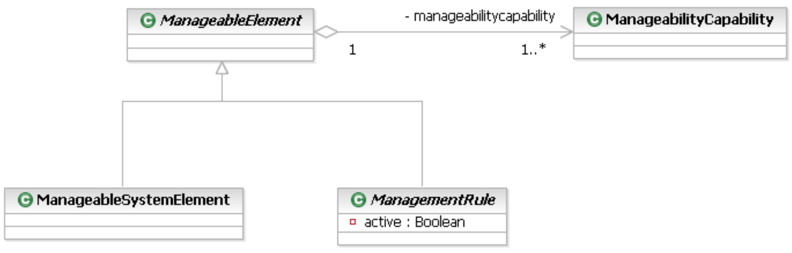

Fig 1: Top level common model for managing systems

From these central concepts, and in line with the work done by the Autonomic Computing Initiative (ACI) [2], we are in the process of evaluating the applicability of the MAPE-K autonomic management control loop. This provides for *monitoring*, data *analysis*, change *planning* and then *execution* of that plan on the basis of static and dynamic system *knowledge*, hence the acronym MAPE-K. We have extended the top-level model above with those concepts that relate specific to this approach.

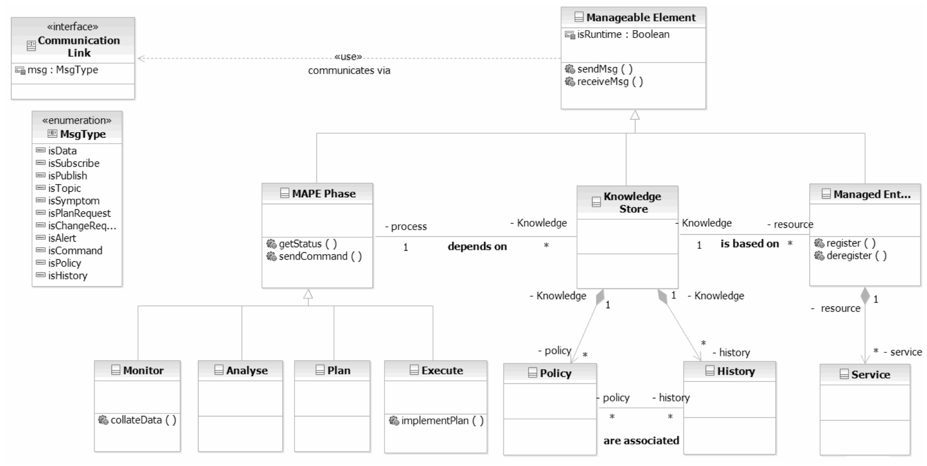

Fig 2: Design-time model of the MAPE-K control loop

Fig 2 illustrates a design-time model of the MAPE-K system management processes. Within the context of the MODELPLEX project, it derives from the common meta-models for system management described above. The MAPE management phases are themselves seen as *ManageableElements* as defined within the common meta-model [3], in much the same way as the *KnowledgeStore* itself containing both static data, or management *Policies*, and dynamic data from the *ManagedEntities*, held as *History*. These objects – *Policies* and *History* – are of particular interest at runtime in providing some way of potentially modifying management behaviours. Our challenge in evaluating such a model for system management lies in how it needs to be implemented, not least to establish whether the data objects can be changed effectively at runtime. This may provide a mechanism for changing the design-time management model. We need to consider further whether other factors need to be examined as well for a truly adaptive management operation.

3 Modelling Management and System Runtime

As far as management modelling is concerned, the MAPE-K loop and the common management meta-models (section 2) provide a conceptual basis. The current challenge is to develop a concrete expression for this framework. This requires defining precisely how models will be used in management, what shape they should take and how they relate to the design process. This section discusses these issues and sketches an experimental system management demonstrator for the monitor and analysis phases that we are implementing. We will begin in this section to consider the initial management phases (monitor and analyze) as they relate to specific aspects of System of Systems (SoS) management.

Enabling model-based system management obviously requires at least two features from the management models. First, since they are used by management operators to observe and analyze system execution, they need to provide a runtime image of the managed system. Beyond that, it should contain management processing information, i.e. define how runtime information is handled (the Monitoring and Analyze phases of the MAPE loop) and – which is beyond the scope of this discussion – how corrective actions are deduced from this information (the Plan and Execute phases of the MAPE loop). On top of these base capabilities, management models should enable more complex management actions, such as determining the root cause of a runtime event (root cause analysis). Ideally, it should also be possible to act not only on the system but also on the management layer itself, i.e. an action on a management rule in the model would result in an actual action on the system management software. Finally, a model-based management tool suite should also allow the consequences of changes on the system or on the management system to be determined before they are performed.

Provided the tool support is adequate, the management concepts we have defined (Fig 3) meet these desired features in theory:

- To begin with, *ManageableElements* represent the monitored system elements. They own collected data (raw runtime data) and indicators (complex monitoring data built from processed collected data). Indicators can be processed to produce Symptoms, which are expressions of a departure from normal SoS function (note that symptoms are not related to any specific *ManageableElement*). Monitoring system execution thus comes down to observing *ManageableElements*, *CollectedData* and Indicators.
- Then, *ManagementRules* contain the management information: *MonitoringRules* express how *CollectedData* is processed to produce Indicators (logical and / or algebraic expressions the operands for which include *CollectedData*) and *AnalysisRules* express how *Indicators* are processed to produce *Symptoms*.

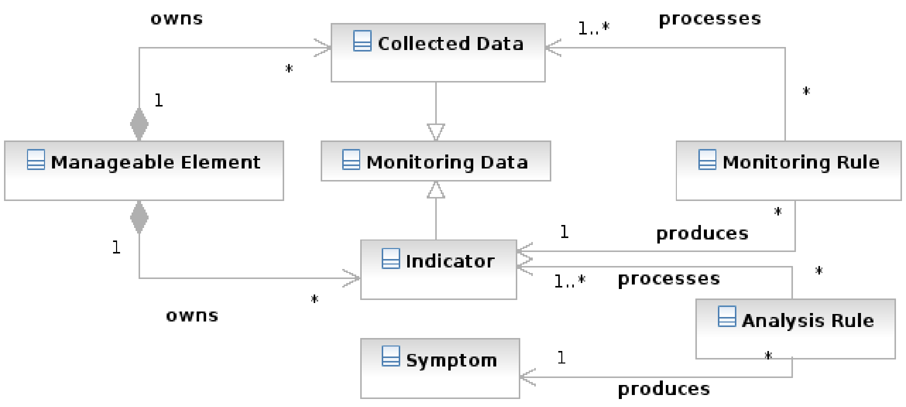

Fig 3: Some of the Management Meta-Model Concepts

However, as usual, concepts need to be put into to practice. In this respect, two concerns prevail: first, the modelling language (i.e. the concrete syntax associated with the meta-models); and subsequently, the tool support. As far as concrete syntax is concerned, it is possible to build runtime models by customizing design models (for instance in UML by creating and using a profile dedicated to management modelling). But this has one major drawback: the resulting models would provide too much detail compared to what is needed for management purposes, and that would reduce readability. We thus chose a different approach. Since we work in the scope of architecture frameworks, we propose to define a specific management viewpoint which will contain the management models. Doing so, these will not be confused with system models for other viewpoints. However, relationships will be defined between them in order to show how *ManageableElements* relate to actual system elements. For the definition of a management modelling language, we considered two equivalent options: either build a UML profile or define a DSL. We had favoured a DSL-based solution in order to enable fast and iterative prototyping.

In order to validate our choices, we have designed an initial experiment: a basic system management prototype which focuses on availability monitoring and analysis. The considered system is a set of cameras whose temperatures and states are monitored. The simulation scenario introduces runtime events (state and temperature changes) from which the monitoring / analysis chain would infer a faulty situation. One of the main outcomes of this prototype was the definition of a DSL for management modelling which gives a concrete shape to the concepts previously described (e.g. symptoms). This DSL was defined in the Microsoft DSL tool, which provides facilities to define the DSL meta-model and the associated concrete syntax and generates the corresponding domain-specific modelling environment. By way of illustration, Fig 4 shows an excerpt of a model of the system described above, built thanks to this DSL. This excerpt represents a *ManageableElement* associated with a camera (*Camera 1 ME*) owning collected data (*StateCD1* and *TempCD1*) and indicators (*S1*, *T1*), and a monitoring rule (*CDFilteringRule1*) which processes *TempCD1* to issue *T1*.

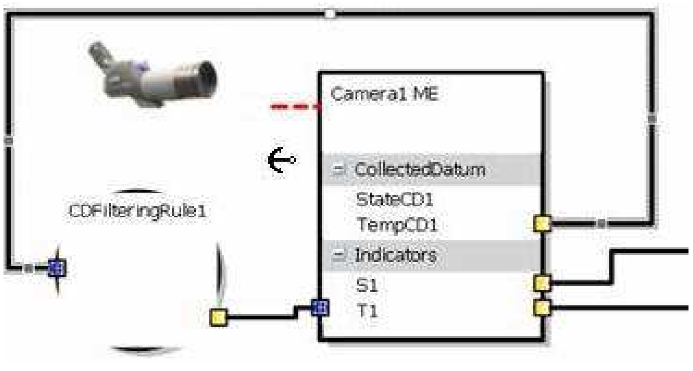

Fig 4: Excerpt of a management DSL model

In order to test our conceptual assets further, we also associated a C# script to each of the monitoring and analysis rules implementing their behaviour. Each time incoming data values change (e.g. in the case of *CDFilteringRule1*, *TempCD1* changes), the script is executed and output data is issued (e.g. *T1* indicator for *CDFiltering1*). Messages are also displayed when events of interest arise (e.g. when a symptom is raised). In this way, the progress of monitoring and analysis can be observed directly on the model. To complete our experiment, a service-oriented Java application was implemented, which simulates the cameras and processes the scenario. This simulation performs dynamic updates of the model in the Microsoft DSL repository, which in turn trigger the model-level monitoring / analysis chain described above. These initial results are quite encouraging, since they demonstrate the global feasibility of our approach on a basic example. We now plan to extend the DSL, in order to enable more complex configurations, e.g. for layered management. We also envisage making some connections with more monitoring infrastructures, like the one described in the next section.

4 Complex Events Processing Architecture

The MAPE-K control loop introduced above provides a useful and extensible design-time model for the semi and fully automatic management of systems or devices. Pickering et al [5] have begun to assess its applicability to highly-distributed devices within a service provider network. In this section, we will focus on the runtime modelling of the monitoring phase specifically.

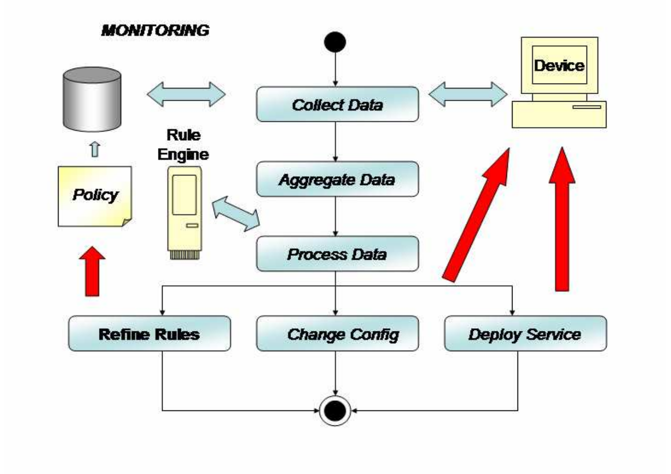

Fig 5: Monitoring Activities at Runtime

Our initial approach to autonomic computing via the MAPE-K framework would suggest a unidirectional process flow for management. Monitoring data are retrieved from the services or devices managed during the initial phase. These are processed and evaluated using policy data from the knowledge store by the analysis and plan phases. The results from these are then sent to the execution phase to effect either are reconfiguration of the managed system or the deployment of a new or modified service. This basic unidirectional flow from monitoring data collection based on policies (*ManagementRules* or service level agreements) to configuration change or service deployment (via the *Execution*) phase is shown in Fig 5.

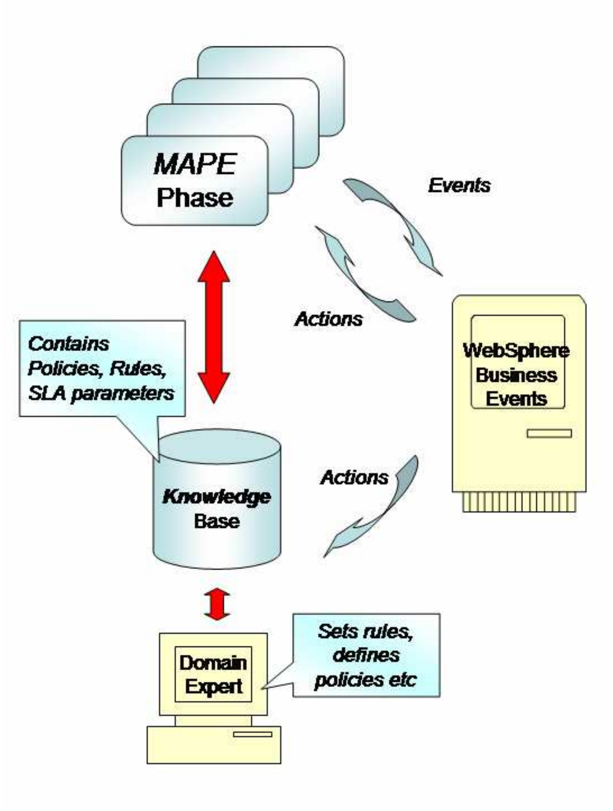
MAPE
Phase
Events
Actions
Contains Policies, Rules, SLA parameters
WebSphere Business Events
Actions
Knowledge Base
Sets rules, defines policies etc
Domain Expert

Fig 6: MAPE-K and Event Processing

The *Policies* can be regarded as the set of domain-specific operations applied to the data obtained by the monitoring process. We use a *rule engine* for complex event processing to aggregate and analyze the data and then make inferences to decide what to do next. The IBM WebSphere® Business Events platform is the best candidate for this rule engine in our architecture since it provides both simple and complex event processing (CEP). We might begin by seeing the rule engine in this case assuming the role of analysis and plan phases in the MAPE-K approach. This preserves the cyclical nature associated with control loops: monitor (or *observe*), decide and take action; then return to the monitor step. In addition to the execution of simple rules related to relatively simple events, such as threshold checking for instance, the rule engine needs to be able to detect complex event patterns in order to provide a complete monitoring process in terms of data aggregation. Complex events may be defined in IBM WebSphere Business Events as rules about the co-occurrence or order of events, but may also be extended to use additional event data for the definition of correlation patterns. With this complex filtering, we do well to reconsider whether the rule engine does fulfill the function of the *Analysis* and *Plan* phases in MAPE-K.

Fig 6 summarises how we use the rule engine within our MAPE-K implementation. Irrespective of MAPE phase, the management process retrieves operational rules (policies, SLA parameters, management rules) from the Knowledge Store at runtime. But also, as data are handled, *MonitoringData* in the case of the monitoring phase, then the rule engine is presented with the data (via an *Event*) to correlate in accordance with the event filters, which as stated may be simple or complex. The result of this processing may result in a change to the operational rules (signaled via an *Action* event to the Knowledge Store); this introduces dynamic data and rule control for our system management model.

In practice, we don't see the sort of uniform behaviour whereby data from a service are monitored, undergo initial processing and are then passed on to the next management phase. Suppose for instance that circumstances change. For instance, rerouting delivery across the network may affect the service provider's ability to meet agreed throughput levels. Such changes will result from the managed system. Equally, some changes may be commercially motivated: customer status or service features may change with a knock-on effect for policy handling. The changes are externally motivated, and independent of the managed system itself. We must therefore allow feedback about the policies and rules from the management system back into the knowledge store from a number of sources. This can be handled as outlined above and summarised in Fig 6: the operational rules are modified by IBM WebSphere Business Events via an *Action* to the Knowledge Store. This in turn may modify how the monitoring is done. As such, the process flow must include a non-device-affecting path back to the knowledge store as well as to the MAPE phases themselves. In Fig 6, *Actions* may therefore return to the MAPE phase as well as to the Knowledge Store. Such *Actions* may be directives (phase I/O parameters or configuration settings), which may include, of course, the next MAPE phase to be called, if any. Our control loop flow is therefore bi-directional. We are able, therefore, to modify how we process the management data from the managed system, but also how the management system itself operates at runtime. Such dynamic changes are reflected back as temporary or permanent modifications to our design-time system management model.

5 Related Works

In section 3, we focused on issues related to runtime system and monitoring operations modeling. This work is based on the common meta-models introduced in section 2 and as such builds on the results of [2] for autonomic management. Our contribution also sits well amidst earlier works on architecture-based system management, like [8], or works about models simulation like [17]. As far as management modelling support is concerned, we are also deriving some benefit from works within domain-specific modelling, such as [16]. Since runtime information layout is one of our concerns, we also have a connection with work such as [15] about dynamic models layouts, though our scope is far more comprehensive. Turning to Section 4, we considered issues pertaining to operational models for system management. Continuing work on autonomic management presented by González *et* *al* [6], we took the ACI MAPE-K framework [2] as our starting point, and more specifically the monitoring phase. Using complex event processing (CEP) [14], we have been able to introduce elements of dynamism to the management system itself; we are now free to generate modified managed system configurations at runtime in contrast to the preloading proposed by Illner *et al* [9] for the service provider domain. In addition, instead of relying on the fixed design-time model of our management control loop, we are also able to introduce changes to the management system itself, and not just adaptation to apply to the managed system along (see [7], 11] and so forth). Further, by viewing management policies, SLAs, and monitoring parameters as dynamic data which can be modified at runtime in response to some CEP-type filtering. Integrating multiple, dynamic data sources of these types introduces the concepts Floch *et al* [7] call for with MADAM.

6 Conclusion

This paper has presented an approach to model-based management for complex systems with a focus on two adjacent aspects. The first is modelling support for management, which entails both model-level visualization of the running system as well as the model-based definition of management functions; and the second is runtime support for complex event processing. On the first aspect, we have proposed a viewpoint dedicated to management concerns. This viewpoint enables – thanks to a dedicated domain-specific language – both monitoring and analysis rules which specify the management logic as well as a runtime view of the system as a set of so-called "manageable elements" to be modelled. This view is then continuously updated as system execution progresses. The second point we deal with relates to runtime management support. We provide an infrastructure based on IBM WebSphere which performs complex and basic runtime event processing (i.e. processes the monitoring and analyze management chain). In accordance with the conceptual model presented in section 2, runtime events can concern the system itself and – something which is not that usual - the management rules themselves. This infrastructure thus permits the terms of management to be modified at runtime.

The added value of our work mainly lies in its comprehensiveness, since we aim at providing support for the whole management chain, from its model-based specification to its realization. Moreover, our strict MDD stance – we clearly place models at the foreground of the development process, both for management specification and system representation at runtime – is not very widespread for such management facilities. Finally, the way we propose to act on the management itself (i.e. to manage the management) at runtime can also be regarded as an original contribution to the field.

On top of implementation and experimentation issues, the next steps will deal with, the improvement of our modelling support. In particular, we plan to enhance the management domain-specific language to enable hierarchical monitoring data processing. This requires in particular the definition of aggregation mechanisms for high-level management indicators (indicators, symptoms), which we have not considered yet. In managing the management system, we are also planning to examine further the implications of distributed management: how to maintain currency or some level of versioning between the original design-time, centralized models and those adapted at runtime; and how and under what circumstances we can distribute complex event processing across the hierarchical network topologies of typical service-providers.

Acknowledgments. The work presented in this paper is being carried out in the context of the MODELPLEX project (IST-FP6-2006 Contract No. 34081), co-funded by the European Commission as part of the 6th Framework Programme.

References

1. Model-based systems management state of the art, MODELPLEX deliverable D5.1.a, MODELPLEX project, 2007.
2. Autonomic Computing, http://www.ibm.com/autonomic/
3. Common meta-models for system management, MODELPLEX deliverable D5.1b, MODELPLEX project, 2007
4. Griffen, C, Huang, R B, Sen, Z, and Fiammante, M “Tranforming UML <<Activity>> Diagrams to WebSphere Business Modeler processes” 2007 http://www.ibm.com/developerworks/websphere/techjournal/0707_fiammante/0707_fiammante.html
5. Pickering, B, Fernández, M A, Castillo, A, Mengusoglu, E “A Domain-Specific Modelling Approach for Autonomic Network Management” 2008 MACE
6. González, J M, Lozano, J A, López de Vergara, J E and Villagrá, V A “Self-adapted Service Offering for Residential Environments” 2007
7. Floch, J, Hallsteinsen, S, Stab,m E, Eliassen, F, Lund, K and Gjørven, E. “Using Architecture Models for Runtime Adaptability” 2006 IEEE Software
8. Garlan, D, Cheng, S-W, Huang, A-C, Schmerl, B and Steenkiste, P “Rainbow: Architecture Based Self-Adaptation with Reusable Infrastructure” 2004 Computer
9. Illner, S, Pohl, A, Krumm, H, Lück, I, Manka, D and Sparenberg, T “Automated Runtime Management of Embedded Service Systems Based on Design-Time Modeling and Model Transformation” 2005 INDIN
10. Kodase, S, Wang, S and Shin, K G “Transforming Structural Model to Runtime Model of Embedded Software with Real-time Constraints” 2003 DATE’03
11. Oreizy, P, Gorlick, M M, Taylor, R N, Heimbigner, D, Johnson, G, Medvidovic, N, Quilici, A, Rosenblum, D.S and Wolf A L “An Architecture-Based Approach to Self-Adaptive Software” 1999 IEEE Intelligent Systems
12. Poirot, P-E, Nogiec, J and Ren, S “A framework for constructing adaptive and reconfigurable systems” 2007 IEEE
13. Ulbrich, A, Weis, T and Geihs, K “QoS Mechanism Composition at Design-Time and Runtime” 2003 ICDCSW’03
14. IBM WebSphere Business Events http://publib.boulder.ibm.com/infocenter/wbevents/v6r1m0/index.jsp
15. S. Prochnow, R. von Hanxleden, "Enhanchements of Statecherts Modeling – the Kiel environment", Artist 2007, Berlin, Germany.
16. Kelly, S., Tolvanen, J.-P., "Domain-Specific Modeling", IEE Computer Society Publications, 2008.
17. Combemale, B., X. Crégut et al., Introducing Simulation and Model Animation in the MDE TopCased Toolkit, ERTS 2008, Toulouse, France.