Middleware for Dynamic Reconfiguration in Distributed Camera Systems

Milan Jovanovic[1] and Bernhard Rinner[2]

[1]Institute for Technical Informatics,
Graz University of Technology, Austria
jovanovic@iti.tugraz.at

[2]Institute of Networked and Embedded Systems,
Klagenfurt University, Austria
bernhard.rinner@uni-klu.ac.at

Abstract — *Networks of embedded smart cameras are an emerging technology for a broad range of pervasive computing applications including smart rooms, intelligent infrastructures and security. Smart cameras combine video sensing, processing and communication within a single embedded device and provide sufficient onboard infrastructure to perform high-level video analysis tasks.*
This paper deals with middleware services required for the efficient deployment and operation of distributed smart cameras. We focus here on services for autonomous and dynamic reconfiguration. Dynamic reconfiguration refers to the exchange of software tasks as well as the alteration of QoS-levels these tasks provide during runtime. Dynamic reconfiguration provides several advantages over statically configured networks including (i) modification of functionality after deployment and during runtime, (ii) adaptation of the network to changes in its internal and external state, and (iii) better exploitation of the available resources.
We have developed the services for dynamic reconfiguration using policies. Policies help to specify rules for the reconfiguration process. By evaluation the policy the new task-level configuration of the network is computed. The reconfiguration is implemented using mobile agents in order to achieve a flexible and scalable middleware service. Our policy-based middleware is demonstrated by a surveillance application.

Keywords: pervasive computing; smart cameras; policy-based middleware; embedded systems.

1 Introduction

Recently much effort has been put into the development of distributed vision systems with smart cameras as key components. Smart cameras are equipped with high-performance onboard computing and communication devices. They combine video sensing, processing and communication within a single embedded device[1]. Networks of distributed smart cameras [2] are an emerging technology for a broad range of important applications including smart rooms, surveillance, tracking and motion analysis. By having access to many views and through cooperation among the individual cameras, these networks have the potential to realize many more complex and challenging applications than single camera systems. Distributed smart cameras are therefore perfect platforms for pervasive computing applications.

The target applications for distributed smart camera systems pose strong requirements on the camera's hardware and software. Typically, the cameras have to execute demanding video processing and compression algorithms. These algorithms executed on the cameras offer different QoS-levels and may be adapted during runtime.

This paper deals with middleware services required for the efficient deployment and operation of distributed smart cameras. This middleware has been implemented on top of our smart cameras [3] and supports the development of distributed camera applications such as traffic surveillance or object tracking. Our smart camera comes with a scalable hardware architecture consisting of a sensing unit, a processing unit and a communication unit. The core components of this embedded platform are a CMOS image sensor, a variable number of digital signal processors (DSPs) for executing various image analysis tasks and a network processor for the camera control. Our smart camera software framework coordinates the various (surveillance) tasks running on (different) processors and supports the data transfer among these tasks using a publisher-subscriber approach [4].

We focus on services for autonomous and dynamic reconfiguration in networks of distributed smart cameras. Dynamic reconfiguration refers to the modification of the functionality in the camera network during runtime. This reconfiguration includes the exchange of software tasks as well as the alteration of QoS-levels these tasks provide. Dynamic reconfiguration provides several advantages over statically configured networks. First, the functionality can be changed during runtime. This allows the introduction of new software tasks even after deployment of the network. Second, the camera network can be better adapted to changes in its internal or external state. Third, by modifying the functionality over time the available resources can be better exploited as well as the availability can be increased.

Although we have developed the middleware services for our networks of smart cameras we feel strongly confident that this work is relevant for many pervasive computing applications. Dynamic reconfiguration is an important network service which helps to promote many advanced applications. Reconfiguration at the task-level is well-established feature in workstation networks. However, not much such work is known on embedded platforms with restricted resources. Our middleware is implemented using mobile agents with focus on flexibility and scalability. Thereby, hardware dependability has been kept rather low which supports the portability to other platforms.

The remainder of this paper is organized as follows. Section 2 briefly discusses related work. Section 3 introduces the dynamic reconfiguration framework and Section 4 describes the implementation. Section 5 explains the taxonomy used to differentiate the dynamic modules used for reconfigurations. Section 6 compares the diverse load-scheduling and shows the results. Section 7 concludes the paper with a brief discussion and a short outlook for future work.

2 Related Work

2.1 Surveillance systems

A modular structure in surveillance systems is not necessary but desirable due to the many heterogeneous technologies built into such a system. Openness and extensibility [5] enable the persistent functionality of the system over a long period of time. In order to enable dynamic reconfiguration in the surveillance system a component-based software architecture is needed. [6] represents one type of component-based software architecture and introduces dynamic reconfiguration based on a multi agent system (MAS). Emphasis of this work is to enable dynamic reconfiguration of exchangeable components and to re-establish connections between them. Dynamic reconfiguration is service-oriented and optional. Due to limited resources in our work, an occasional dynamic reconfiguration is essential and it takes QoS with emphasis on load balancing into account.

2.2 Policy-based middleware

The *policy-based middleware* is a result of recent research in the area of network management automation. A noticeable work in this field has been done by IBM research center. The IBM team has developed an own policy language and has evaluated it with new policy-framework on a storage area network (SAN) [7].

IBM Research has also developed the *policy toolkit* [8], which has the goal to integrate a policy-based structure in different applications. This toolkit supports different types of policy languages. In order to enable a policy-based system the user has to select the syntax of the policy language, modules and pattern.

Several benefits of policy-based middleware have been explained by [9]. First, this approach is advantageous when the system works in advance in unspecified environment. Second, a policy-based approach supports different system's behaviors corresponding to different situations of the environment. Finally, it supports upgrading. Easy system upgrade is only possible if it has a modular structure with well defined and open interfaces. [9] also presents a novel agent framework with integrated policy control which is used for wireless ad hoc network management.

Policy-Enabled Mobile Application (Poema) [10] is a policy-based middleware which supports code mobility and supports different strategies including MAS for a reconfiguration tasks. In Poema, mobility strategies can be changed even during application execution.

Policy middleware as paradigm is supported from the Internet Engineering Task Force (IETF) and Distributed Management Task Force (DMTF) with many standards from the information models ([11], [12], [13]), over the policy framework design [14], to the policy communication protocols ([15], [16]).

[17] introduces policy-based management environment for mobile users. This work combines the different profiles with policy-enabled agents to tailor the services according to users preferences and the network capabilities. This approach differs between the user-policies, network-policies which are utilized for configuration of network devices, and agent-policies which control the agent behavior.

The policy management architecture introduced in [18] deals with the problem of multiple policy enforcement points (PEP) [19]. The policy framework in this approach allows

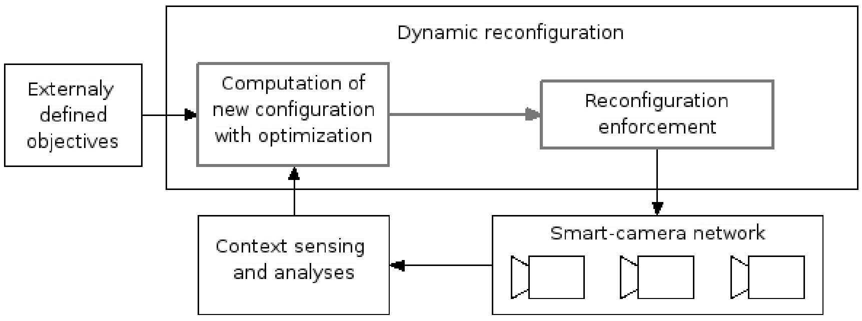
Dynamic reconfiguration
Externaly defined objectives
Computation of new configuration with optimization
Reconfiguration enforcement
Smart-camera network
Context sensing and analyses

Figure 1: Dynamic reconfiguration loop.

that policies can be specified at a higher level and mapped into an appropriate *local rule-set* by using an automated tool. This work is based on policy language Houdini which has been developed at Bell Labs.

A more abstract approach for policy oriented management is shown in [20] and [21] where a general purpose ontology language (OWL) is used to define the network policy management information. Advantages of this approach, described in [20], are simplified information maintenance due to unified definition of elements and their behavior. A second advantage lies in the semantic expressiveness of ontology languages and already existing tools for their validation. In [22] policies are used for negotiation in MAS environment. The MAS uses policies expressed in XML for negotiation in order to enable service composition and cooperation.

3 Dynamic Reconfiguration of the Network

3.1 Configuration Approach

Dynamic reconfiguration refers to the modification of software tasks executed on nodes of the camera network as well as the alteration of the QoS-level provided by these tasks. More formally, a *configuration* c_i of the network is defined as as tuple in the *configuration space*

$$C = (s \times q_s \times r) \tag{1}$$

where s refers to the available services (or tasks) in the network, q_s refers to the service levels of the tasks, and r refers the resources where the tasks can be executed. The available processors represent the resources in the network. A reconfiguration corresponds to a transition from one tuple to another $(c_i \rightarrow c_j)$ in the configuration space.

The overall objective of dynamic reconfiguration is to transfer the network into a specific configuration which better captures the current requirements. In order to achieve such an improvement some form of reasoning about the specific configurations is required. Thus, we need some information about the state of the current configuration as well as some objective for the next configuration.

”Figure 1” depicts the reconfiguration loop of the camera network. The main steps of this loop can be described as follows:

Context sensing and analysis The goal of context sensing and analysis is to gather data from the network and to deliver contextual information. The context includes information about the internal and external states of the network. The internal state can be composed by the actual configuration and the available resources. Events detected by the tasks, such as a recognized person or a successful object tracking can be assigned to the external state. Contextual information is strongly dependent on the application. We, therefore, do not go into more detail here about this step.

Computation of new configuration with optimization This step determines the next configuration of the network. In order to reason about configurations this step requires some model about the network. The contextual information of the network and some objective serve as input for this reasoning. As described in the following section we use a policy-based approach for the optimization.

Reconfiguration enforcement The final step in the reconfiguration loop performs the actual transfer to the new configuration. This step needs, therefore, mechanisms to alter the QoS-level as well as to modify the mapping of tasks onto resources. In order to perform dynamic reconfiguration we have adopted mobile agents and developed a middleware framework on our network nodes.

3.2 Policy-based Framework

The computation of a new configuration is based on policies. Policies are distributed within the smart camera network and this policy distribution enables the distributed decision-making process for the reconfiguration of the smart camera network. Policies represent a collection of rules which have to be calculated to achieve a potential new configuration. Thus, our policy-based approach can be separated into a policy-specification and a policy evaluation phase. As depicted in ”Figure 1” the policies are externally specified and serve as input to the reconfiguration loop. The evaluation of the policies has to be performed within the loop. In order to get some directives from the policy-framework, the policies have to be evaluated. This evaluation is performed in a hierarchical way, from higher abstract objects (policies) over rules to conditions and actions. This evaluation chain is triggered whenever one event occurs and changes the corresponding parameters in policy.

3.3 Policy information model

In order to enable the policy based management, the target system has to be modeled like a state machine, where one state corresponds to one configuration of the camera network. The policies are used to manage the transformation of the target system from one state (configuration) to another.

The object-oriented information model for representing policies is in our project constructed using Policy Common Information Model (PCIM) [12] as paradigm and the PCIM extensions [13] as well. According to the specific construction of our platform and limitations in an embedded environment, only necessary features from those standards are implemented. The core of the policy-based structure consists of: Policy, Rule, Condition, Action and Parameter object. This set of objects has a common inherited iden-

Name	**Definition**
Policy	$\mathbf{P} = \mathbf{R} \mid \mathbf{P} \{ , \mathbf{R} \mid \mathbf{P} \}$
Rule	$\mathbf{R} = [\mathbf{C}] , \mathbf{A}$
Condition	$\mathbf{C} = \mathbf{Par} \mid \mathbf{C} , \mathbf{op} \{ , \mathbf{Par} \mid \mathbf{C} \}$
Action	$\mathbf{A} = \mathbf{Par} \mid \mathbf{A} , \mathbf{ba} \{ , \mathbf{Par} \mid \mathbf{A} \}$
Parameter	$\mathbf{Par} = \mathbf{Integer} \mid \mathbf{Long} \mid \mathbf{Boolean} \mid \mathbf{String} \mid \mathbf{Object}$
Logical Operator	$\mathbf{op} = \mathbf{Integer} \mid \mathbf{Char} \mid \mathbf{String}$
Basic Action ID	$\mathbf{ba} = \mathbf{Integer}$

Table 1: Definition of our Policy structure

tification part that uniquely identifies the object. In our policy model as shown in ”Table 1”, a policy is structured as a tree. The policy consists of sub-policies and rules.

Combining several sub-policies into one builds a logical unit with greater capabilities. This grouping of policies also safes memory resources and simplifies the construction of policies.

Policies can be changed in real time and enable different execution flows during the execution of scenarios. A *scenario* describes a successive reconfiguration of the smart camera platform from one state to another. This dynamic structure of a policy is considered as a possibility that the sub-policies can be enabled or disabled within the scope of a policy grouping.

A rule is an entity that encapsulates conditions and actions. If a condition is satisfied the corresponding action or list of actions will be executed. The execution order of the permitted actions is arbitrary. The condition object is responsible for the decision-making process and contains parameters and sub-conditions. Number and type of these objects depend on the nature of the operator, which is defined in that particular object. However, the return value from the computation of a condition is a boolean value and can be compared only with corresponding boolean values and proper logical operators. Parameter type checking is done through the processing of conditions. The processing of conditions is a recursive procedure for all sub-conditions as well.

Actions can be initialized as single (basic) action or as collection of actions (sub-actions). A basic action is defined with an unique identification and a collection of parameters. This identification number has to be defined in the configuration layer (see Section 4.1) and the corresponding action also has to be implemented in the executing layer (see Section 4.3) in order to enable that this action can be mapped in the policy layer (see Section 4.2).

A parameter is an object that wraps the primitive data types or another object. Parameters are placed as leafs in the condition tree, but they are also contained as parameters in an action object. Types of parameters are divided into *atomic type* (boolean, integer, long, string) and *object type*. This wrapping is needed in order to identify the parameters in distributed environment and to facilitate the access to its value.

POLICY 1	Initialize and start motion detection
RULE 1	If motion detection agent not exist create motion detection agent
CONDITION 1	If motion detection agent not exist
OPERATOR (=)	
PARAMETER 1 (pMDagent)	Pointer to motion detection agent with NULL as default value
PARAMETER 2 (NULL)	Constant with NULL value
ACTION 1	Create motion detection agent
BASIC ACTION ID (1)	
PARAMETER 3 (ClusterName)	Target cluster
PARAMETER 4 (NodeID)	Target node
PARAMETER 5 (MDagent)	Agent type
PARAMETER 1 (pMDagent)	Return value: Pointer to new created agent
RULE 2	If motion agent is ready load the algorithm
CONDITION 2	If motion detection agent exist and algorithm is not loaded
OPERATOR (AND)	
CONDITION 1	If motion detection agent exist(this condition is the same as in RULE1)
...	
CONDITION 3	If motion detection algorithm is not loaded
OPERATOR (=)	
PARAMETER 6 (AlgorithmLoaded)	Boolean variable with FALSE as default value
PARAMETER 7 (FALSE)	Constant with FALSE as value
ACTION 2	Load the algorithm
BASIC ACTION ID (2)	
PARAMETER 3 (ClusterName)	Target cluster
PARAMETER 1 (pMDagent)	Pointer on motion detection agent with NULL as default value
...	
PARAMETER 6 (AlgorithmLoaded)	Boolean variable with FALSE as default value
RULE 3	If motion detection algorithm is loaded start the algorithm
CONDITION 4	If motion detection algorithm is loaded
OPERATOR (!=)	
PARAMETER 6 (AlgorithmLoaded)	Boolean variable with FALSE as default value
PARAMETER 7 (FALSE)	Constant with FALSE as value
ACTION 3	Start the algorithm
BASIC ACTION ID (3)	
PARAMETER 3 (ClusterName)	Target cluster
PARAMETER 1 (pMDagent)	Pointer to new created agent
...	
PARAMETER 9 (MotionEvent)	Motion event variable with FALSE as default value

Table 2: Simplified example of motion-detection policy

3.4 Reconfiguration enforcement

After the evaluation of policies, the resulting list of actions with corresponding parameters have to be enforced and the system is transferred to a new state. Real task allocation in the system can be achieved in two ways. The first way is with the direct call of necessary encapsulated basic actions, which means that all decisions for task distribution are already met. The second way is over the algorithm for CSP (Constrained Satisfaction Problem) [23], which has already integrated all basic actions necessary for reconfiguration.

3.5 Evaluation of the reconfiguring scenarios

The evaluation of our policy-based framework is done with distinct defined policies that determine the various flows of reconfigurations. A simplified meta-presentation is depicted in "Table 2". This example shows a meta-XML presentation of a motion detection policy. The aim of this policy is to define a usual surveillance task. This policy consists of three rules. All rules are processed in one pass through the policy. Conditions in this case are specified so that after the first processing of the policy only *Action 1* can be executed. After execution of *Action 1* and updating of the corresponding parameters *Action 2* will be in queue for execution. The *Action 2* is the loading of *dynamic module* (see Section 5) and represents the essential functionality (see Section 6) in the reconfiguration process. After execution of *Action 2* and again updating, the condition for *Action 3* will be satisfied. This type of condition specification enables the sequential execution of actions with a strict sequence order. In case of an alternate specification of conditions, more that one action can be send to the enforcement unit for execution. After all three actions are executed the SmartCam is ready to detect motion-events. The presented policy for motion detection can be used as template and combined with another policies (object recognition, tracking ...). Combining more policy-templates user can easy build new behavior of the surveillance system without knowing the all specific details.

4 Implementation

The overall objective of our project is to enable the autonomous dynamic reconfiguration in a distributed environment such as a SmartCam network. This reconfiguration refers to the supplying of the underlying resources, which in the SmartCam case are DSP processors, with versatile software. The SmartCam consists of one network processor and an arbitrary number of DSPs which communicate between themselves over the PCI bus. Our framework consists of the following functional units:

4.1 Configuration layer

This layer represents the graphical user interface that simplifies the process of defining the policies as templates for executing scenarios. The policies are stored in XML files as convenient format for tree like data structures. Every policy object can be serialized in XML format and vice versa. In order to be used by agents, policies are first parsed from XML into Java objects with the SAX parser.

4.2 Policy layer

The policy layer is the logical part of the application and the most complex module. In order to enable storing of policies from object representation, XML is chosen as best suitable for that purpose. The core policy classes are the kernel of this layer. Every object from this set has a serialization function that transforms an object into XML form. To speed up work with internal parameters, pointers of the all internal parameters in policy are stored in a list. Using this parameter list, maintaining the parameter's consistency is faster and easier. A policy object has the possibility to be active or inactive. This feature enables in real time the fast changing of policy behavior.

4.3 Executing layer

To enforce the reconfiguration of a system, our policy middleware has an executing layer which is aware of all basic actions defined in configuration layer. This unit contains a list of basic actions that have to be executed without delay. The current policy middleware implementation is based on [24] [23] as a basic framework. All function calls important for reconfiguration are extracted in this layer and encapsulated as basic actions. Now all these basic actions are uniquely identified and offered as available services to the configuration layer.

5 Dynamic Modules

The image processing modules are the typical piece of software which are exchanged at runtime. These modules represent the diverse algorithms which can be executed on the DSP processors. "Figure 2" depicts the dynamic module with corresponding connections and communication path with MAS. All dynamic modules have the common interface

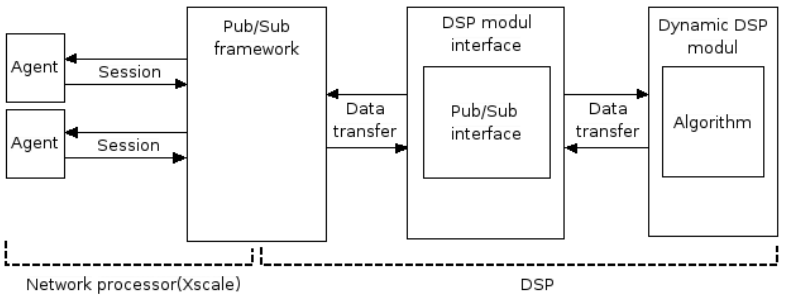

Figure 2: DSP dynamic module in SmartCam environment.

which enables the connection with the DSP framework. The dynamic modules communicate with each other or with java agents over the publisher/subscriber (PS) framework[4]. The connection between the java agents and the PS framework is enabled through the java native interface. The java agent has one connection with the PS framework and one such connection has one or more PS sessions for communication with a certain task on DSP. One PS session usually contains one publisher and one subscriber on both sides of the agent-DSP communication. Over the publisher the agent initialize the DSP task and sends the control messages. The subscriber on the agent side is used to receive the processed data from the DSP task. The dynamic modules have also their own publisher and subscriber for the corresponding session with the agents.
Available modules that we have used for testing are divided into 3 groups and each group of modules represents one step in the surveillance scenarios.

1. The first group of modules are motion detectors. As usually every surveillance scenario starts with a motion detection. These modules use different algorithms depending on the background model they use to distinguish between a (moving) object and the background. Every algorithm has different memory and processing requirements as depicted in ”Table 3”.

Algorithm	Processing time [processor cycles]	Required memory[kB]
Temporal Differencing	5.2 x 10^6	198
Kalman filter	8 x 10^6	297
Single Gaussian	10 x 10^6	396

Table 3: Resource requirements for motion detectors

2. The second group are object recognizers and classifiers. One such module receives the information about a motion detection event, as earlier described, and classifies the object and forwards the information further. This step in a surveillance scenario represents the branching in case of the recognition of more than one object.

3. The third group are trackers. This module receives the information about the current position of object which should be followed and continues to follow the object. The tracker runs first local on the same camera and when object lives the surveilled area

of the first camera continues the following on the next camera which is connected with previous one with overlap surveilled areas [25].

6 Dynamic Loading

Dynamic loading in the *policy middleware* considers runtime exchange of the executable software on the DSP processors. Dynamic loading is initiated for the dynamic modules which have to be loaded to DSPs according to the new configuration. The dynamic modules which don't have to be removed from DSPs as a result of reconfiguration remain active and enable the SmartCam to be operational during the reconfiguration process. In the SmartCam project the DSPs are the main focus of resources for dynamic reconfiguration by an autonomous resource management. In our case study for the reconfiguration methods, we differentiate the following scheduling approaches:

- Case 1: The first case focuses on the time required for initialization of a new module. The new modules are in standby state and wait to be activated in order to take over data processing. After the prior module finishes its task corresponding data are sent to the next module. This case considers the time of reaction and presumes enough available memory for the standby tasks.
- Case 2: The second case considers the time for loading and for initialization of a new module. The new module in the surveillance scenario is loaded after an event from the running module. This case considers the situation when the type of the new module which has to be loaded is not known in advance or the number of instances depends on the results from the previous module (Example: after the recognition and classification of multi objects, as many trackers as objects has been recognized will be loaded on DSP). This case does not consider the memory limitations, i.e. the new module fits into the available memory.
- Case 3: The third case considers a memory deficiency. The loading and the initialization of the new module in a surveillance scenario have to wait until the previous module has been finished and unloaded from the DSP as well. This case considers the memory availability as bottleneck and presumes a memory deficiency for the standby tasks.

"Table 4" depicts the measurement for different scheduling cases which represent the time needed for an activation of the dynamic module. This module activation represents the time interval from the moment when the previous task finishes his job $(T_{EnforcementStart})$ till the new task is ready for the execution $(T_{EnforcementEnd})$. The measurements are based on modules with 15kB file size.

Scheduling case	$T_{EnforcementEnd} - T_{EnforcementStart} [ms]$				
Case 1	166	201	198	200	199
Case 2	255	246	226	215	192
Case 3	425	412	397	381	360

Table 4: Activation times of dynamic modules for the different scheduling cases.

”Table 5” shows the dependency between the size of the dynamic modules and the time needed for loading on the DSP. The intercommunication with agent and module initialization are not encountered.

Module size	$T_{LoadingEnd} - T_{LoadingStart} [ms]$				
5.6 kB	48	45	43	41	42
10 kB	55	58	49	55	58
262 kB	165	167	177	165	170

Table 5: Loading times of dynamic modules dependent on the module size.

7 Conclusion

This paper has described the policy-based middleware for distributed embedded systems. A policy controlled MAS provides the autonomous behavior and overall control of the system. This includes the real-time dynamic reconfiguration as main goal in our project. Limited resources in the SmartCam embedded platform determine the specific construction of the policy-based middleware with only preferred features. The results depicted in this paper are intermediate steps in the overall reconfiguration process. Future steps in our work will focus on the evaluation of this framework on the next generation of embedded surveillance systems, a further optimization of the reconfiguring methods and including other policy standards to support heterogeneous platforms.

Acknowledgments

This work has taken place at Graz University of Technology, and has been supported by the Austrian Research Promotion Agency.

References

[1] Wayne Wolf, Burak Ozer, and Tiehan Lv. Smart Cameras as Embedded Systems. *Computer*, 35(9):48–53, September 2002.

[2] Bernhard Rinner and Wayne Wolf, editors. *Proceedings of the Workshop of Distributed Smart Cameras (DSC-06)*, Boulder, CO, USA, October 2006.

[3] Michael Bramberger, Andreas Doblander, Arnold Maier, Bernhard Rinner, and Helmut Schwabach. Distributed Embedded Smart Cameras for Surveillance Applications. *Computer*, 39(2):68–75, February 2006.

[4] Andreas Doblander, Bernhard Rinner, Norbert Trenkwalder, and Andreas Zoufal. A light-weight publisher-subscriber middleware for dynamic reconfiguration in networks of embedded smart cameras. In *Proceedings of the 5th WSEAS International Conference on Software Engineering,Parallel and Distributed Systems (SEPADS’06)*, feb 2006.

[5] Chiao-Fe Shu, Hampapur A., Lu M., Brown L., J. Connell, Senior A., and Yingli Tian. Ibm smart surveillance system (s3): a open and extensible framework for event based surveillance. In *Proceedings of the IEEE Conference on Advanced Video and Signal Based Surveillance*, pages 318 – 323, September 2005.

[6] Karim Djouani Yaicne Narsis, Yacine Amirat. Dynamic reconfiguration service for component based software architecture. In *Proceedings of the 2004 IEEE Conference on Cybernetics and Inteligent Systems*, pages 23 – 28 vol.1, Singapore, December 2004.

[7] D. Agrawal, S. Calo, J. Giles, Kang-Won Lee, and D. Verma. Policy management for networked systems and applications. In *9th IFIP/IEEE International Symposium on Integrated Network Management, 2005*, pages 455 – 468, 15-19 May 2005.

[8] Dinesh C.Verma and Seraphin B.Calo. A toolkit for policy enablement in autonomic computing. In *Proceedings of the International Conference on Autonomic Computing (ICAC04)*, pages 270 – 271, May 2004.

[9] Chiang C.-Y.J., Chadha R., Yuu-Heng Cheng, Levin G., Shihwei Li, and Poylisher A. A novel software agent framework with embedded policy control. In *Military Communications Conference MILCOM 2005. IEEE*, pages 2863 – 2869 Vol. 5, October 2005.

[10] Montanari R., Lupu E., and Stefanelli C. Policy-based dynamic reconfiguration of mobile-code applications. *Computer*, 37(7):73 – 80, july 2004.

[11] Distributed Management Task Force (DMTF). Common information model (cim); http://www.dmtf.org/standards/cim/.

[12] B. Moore, E. Ellesson, J. Strassner, and A. Westerinen. Policy Core Information Model – Version 1 Specification. RFC 3060 (Proposed Standard), February 2001. Updated by RFC 3460.

[13] B. Moore. Policy Core Information Model (PCIM) Extensions. RFC 3460 (Proposed Standard), January 2003.

[14] R. Yavatkar, D. Pendarakis, and R. Guerin. A framework for policy-based admission control. RFC 2753 (Informational), January 2000.

[15] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, and A. Sastry. The cops (common open policy service) protocol. RFC 2748 (Proposed Standard), January 2000. Updated by RFC 4261.

[16] K. Chan, J. Seligson, D. Durham, S. Gai, K. McCloghrie, S. Herzog, F. Reichmeyer, R. Yavatkar, and A. Smith. Cops usage for policy provisioning (cops-pr). RFC 3084 (Proposed Standard), March 2001.

[17] M. Ganna and E. Horlait. On using policies for managing service provisioning in agent-based heterogenous environments for mobile user. In *Proceedings of Sixth IEEE International Workshop on Policies for Distributed Systems and Networks(POLICY'05)*, pages 149 – 158, june 2005.

[18] Richard Hull, Bharat Kumar, and Daniel Lieuwen. Towards federated policy management. In *Proceedings of the 4th International Workshop on Policies for Distributed Systems and Networks (POLICY03)*, pages 183 – 194, 2003.

[19] A. Westerinen, J. Schnizlein, J. Strassner, M. Scherling, B. Quinn, S. Herzog, A. Huynh, M. Carlson, J. Perry, and S. Waldbusser. Terminology for Policy-Based Management. RFC 3198 (Informational), November 2001.

[20] Chamoun M., Kilany R., and Serhrouchni A. Proposition of a network policy management ontology. In *Proceedings of the Fourth IEEE International Symposium on Signal Processing and Information Technology*, pages 262 – 266, December 2004.

[21] Maroun CHAMOUN, Rima KILANY, and Ahmed SERHROUCHNI. A semantic active policy-based management architecture. In *Proceedings of IEEE Workshop on IP Operations and Management*, pages 224 – 232, October 2004.

[22] Jing-Fan Tang and Xiao-Liang Xu. An adaptive model of service composition based on policy driven and multi-agent negotiation. In *Proceedings of Fifth International Conference on Machine Learning and Cybernetics*, pages 113 – 118, August 2006.

[23] Thomas Winkler. Load distribution for embedded smart cameras based on mobile agents. Master's thesis, TU-Graz, May 2005.

[24] Markus Quaritsch. An agent-based framework for object. Master's thesis, TU-Graz, May 2005.

[25] Markus Quaritsch, Markus Kreuzthaler, Bernhard Rinner, Horst Bischof, and Bernhard Strobl. Autonomous multi-camera tracking on embedded smart cameras. *EURASIP Journal on Embedded Systems*, 2007.