Extending the MaSE Methodology for the Development of Embedded Real-Time Systems

Iman Badr[1], Hisham Mubarak[1], and Peter Göhner[1]

[1] Universität Stuttgart, Institute of Industrial Automation and Software Engineering (IAS),
Pfaffenwaldring 47
70550 Stuttgart, Germany
{Iman.Badr, Hisham.Mubarak, Peter.Goehner}@ias.uni-stuttgart.de

Abstract. Embedded real-time systems play an important role in various application areas like plant automation, product automation or car electronics. In recent years a considerable growth in the functionality has been observed. At the same time, expectations on systems' flexibility at runtime are growing steadily. The paradigm of agent-oriented software engineering is a well suited approach for the development of decentralised, complex software systems with high flexibility. A number of software engineering methodologies have been introduced for developing agent oriented systems. However, none of the existing methodologies is intended for the development of embedded real-time systems. This work presents an extension to the Multi-agent Systems Engineering (MaSE) methodology that tackles domain-specific weaknesses and defines a systematic procedure for the development of agent-oriented embedded real-time systems.

Keywords: Agent-Oriented Software Development, Real-Time Systems, Embedded Systems

1. Introduction

Embedded systems are systems that are integrated logically and physically in a device or a larger system. Their application spectrum ranges from simple devices like mobile phones and house held devices up to the complex ones like aircrafts and industrial process controllers, to name a few. Regardless of the diversity of their application domain, all embedded systems are required to synchronise their execution with the technical process of the encapsulating device. Traditionally, the development of real-time systems was targeted for closed predefined hardware structures. More and more the structure of systems hardware becomes dynamic with the addition, removal and upgrade of components. Current systems are thus required to adapt to dynamic changes in the structure of the hardware as well as to flexibly deal with unforeseeable events that may occur in the working environment. Therefore, the development of such systems represents a challenge of achieving flexibility without violating

invariant requirements, especially real-time requirements which are considered crucial for these systems.

With their special nature in tackling the complexity of distributed applications and adapting their behaviour to stochastic, dynamically changing environments, software agents represent a suitable approach for developing flexible embedded systems. Such development is supposed to be based on and guided by systematic methodologies that result in exhibiting the required controlled flexibility. However, the appealing concepts of agents are not complemented with powerful comprehensive methodologies that provide the needed support along the different application domains. Concerning the embedded real-time systems domain, the available agent-oriented methodologies provide no support for indispensable features namely, timeliness and concurrency. Therefore, the employment of agents in real-time systems may lead to possible violations of the timeliness requirements. Consequently, the potential of the agent-based paradigm in dealing with complexity and adapting to dynamic conditions has not been utilised in the embedded systems field.

In order to pave the way for the employment of agents in the embedded real-time systems domain and capitalise on their flexibility, this work aims at bridging the gap between the embedded systems domain and the available agent oriented software engineering. Towards achieving this objective, the Multi-agent Software Engineering (MaSE) methodology [1] was extended with timeliness concepts that customise it for the embedded systems domain. .

This paper is organised as follows. Section 2 overviews the special requirements and conventional development trends of embedded systems. Section 3 focuses on the MaSE methodology by first reviewing the comparative study that led to its selection and then discussing its limitations in capturing the special characteristics of embedded systems. Section 4 proposes a set of extensions that overcome these limitations. Section 5 illustrates the proposed extensions with a case study. Section 6 presents concluding remarks and an outlook on future work.

2. Embedded Systems

Unlike information systems whose development is targeted basically to the satisfaction of the customer needs, embedded systems have to satisfy the goals and desires of the customer while at the same time complying with the requirements and constraints enforced by its controlled process. To illustrate, consider an embedded system of a typical automatically controlled washing machine. Such system is expected to provide a user interface controlling the required washing program. In addition, it is obliged to react on the right time to the events continually emerging from the technical washing process like an event signaling the fall of the water level under a certain threshold. Failing to react to such events on the right time may cause undesired effects or for safety critical systems may bring about dangerous or even fatal effects.

2.1. Distinguishing Characteristics

In light of the previously mentioned example, two basic distinguishing characteristics of embedded systems could be identified

2.1.1. Timeliness

To synchronise their operation with the controlled physical process, embedded systems are required to work under timing constraints that stem basically from the technical system. In other words, embedded systems are real-time systems whose input, processing and output have to be performed under predefined timing requirements [2]. Timing requirements usually result from the physical laws governing the controlled technical process. For example, an automobile engine system controls the amount of proper fuel to be injected into the combustion chamber of each cylinder. For such an example, a delay in terms of a few microseconds may lead to opening the valve at an incorrect point of time which results in the mechanical damage of the engine [3]. This example demonstrates how significant the timeliness requirements of real-time systems could be. In general, real-time systems are classified according to the nature of their real time requirements into soft and hard real-time systems. While soft real-time systems work under relatively flexible timing constraints which when violated can lead to lowering the performance but can still be tolerated, hard real-time systems have much more strict constraints whose violation can lead to a failure or can be dangerous.

2.1.2. Concurrency

Embedded real time systems are concurrent by nature in that they have to react to several sensors and control multiple actuators simultaneously to achieve the required performance on the right time. Concurrency raises several challenges like scheduling, synchronisation, and communication of tasks. Tasks are either executed periodically or are triggered by events whose occurrence time is not determined a priori. Each task works under timing constraints and has to meet a certain deadline [2]. The objective of a real-time system is to satisfy the requests of all tasks in a way that all deadlines can be met. However, due to limited resources, this is not always possible. Therefore, priorities of tasks have to be considered in scheduling to make sure that time critical tasks with hard deadlines are not delayed. In case embedded real-time systems are distributed the problem becomes more complex with the need to synchronise the execution of parallel tasks in addition to synchronising the clock of all nodes [3].

2.2. Conventional Development Trends

The engineering trends of embedded systems have featured major changes all over the years. During its early stages, embedded systems were developed in an ad-hoc manner, where the system was realised by engineers having little knowledge of computer science. They tended to satisfy the requirements at hand by sketching a block diagram of the system to be implemented with special considerations to saving hardware resources at the expense of the software capacity. Software was just limited

to stand-alone implementation running on a microcontroller with no operating system. With increasing market needs, more attention was given to adding software functionalities to enhance the utilisation of the system which resulted in an increasing complexity of the software [3]. Currently, the industrial trend is characterised by designing embedded systems with in-house methods that are specifically tailored to their application domain. In general, a co-design approach is adopted as a natural model for conceiving the strong interrelation between hardware and software [4].

2.3. Modelling Techniques

Due to their inherent complexity, embedded real-time systems depend heavily on computational models for their analysis and design. They serve in formally specifying the temporal and concurrent aspects of the behaviour of the system in an unambiguous manner that simplifies implementation and testing. In general, real-time systems are usually modelled by state-oriented models that stress the control and reactive aspects of the system by capturing the effect of the external events coming from the environment on the states of the system. These models attach special importance to temporal and concurrency issues. Out of the existing models, finite state machines and Petri-nets are most commonly used for modelling the behaviour of embedded systems [5].

2.4. Flexibility Requirement

The embedded systems industry is featuring exponential growth motivated by the increasing availability of cheaper and more powerful hardware components. Due to the relatively long lifetime of devices incorporating embedded systems, it is highly demanding to design an embedded system in such a way that modifications to the hardware structure by adding or removing components should have no or minimal effects on the existing software. Consequently, the conventional approach of designing closed software which is tightly coupled with the underlying hardware components does not offer sufficient flexibility in face of the new challenges. This motivates investigating new approaches of software engineering like the agent-oriented approach which deals with uncertainty in the working environment and complexity in behaviour and functionality by designing autonomous and communicating entities called software agents. Agent oriented software engineering is a good approach for distributed, ill-structured and dynamic systems [6].

3. The MaSE Methodology

Mulit-agent Systems Engineering (MaSE) is a generic agent-oriented software development methodology [1]. The engineering process of MaSE is based on a top-down software engineering approach that supports the analysis and design phases through seven steps, which are be performed in an iterative fashion. The whole process with the steps and the corresponding artefacts is depicted by Fig. 1. As

illustrated in the figure, MaSE adopts a goal-oriented analysis by deriving the system goals from a set of system requirements – whose generation is assumed to be outside the scope of MaSE. The system is next modelled as a set of roles which are assigned the identified goals. During the design phase, the identified roles are grouped together to form agents that are designed to play the incorporated roles. The design phase extends up to the deployment stage, where a decision on the distribution of agents to the available physical platforms is taken.

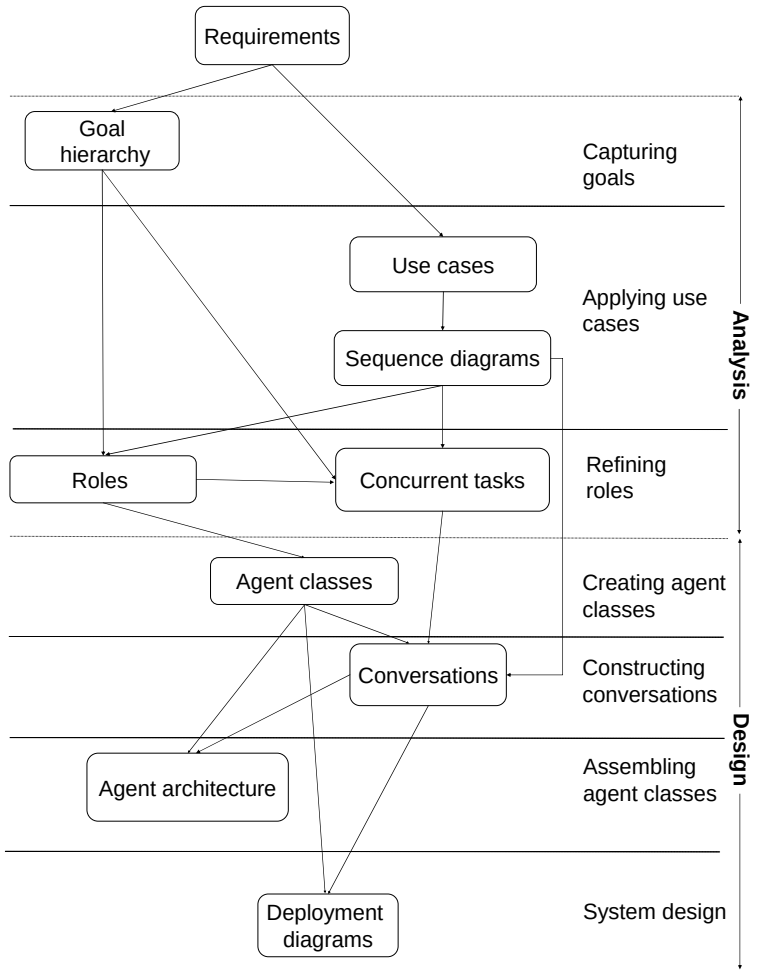

Fig. 1. The software engineering process of MaSE [1]

3.1. The Selection of MaSE

This work builds on a previous study of agent-oriented development methodologies that resulted in the selection of MaSE as the agent-oriented methodology with the best

relative potential for the embedded systems domain [7]. During the course of this work, a two-phase evaluation process was conducted.

The first phase served in deciding on an initial set of methodologies that represent good candidates for further deeper investigation out of the excessive number of the currently available methodologies that amount to twenty different methodologies [8]. The evaluation criteria for this phase considered broad aspects like the soundness of concepts, the suitability to the embedded systems domain, the coverage of the development process, and the tool support. This initial evaluation resulted in the selection of Gaia [9], MaSE [10], Prometheus [11], and PASSI [12].

During the second phase, the four methodologies were assessed against a framework of attributes according to the well-known feature-based evaluation method. The evaluation criteria covered twenty seven different features grouped into five categories that examine the support of the methodologies to aspects related to the application domain, the development process, the agent-oriented features, the system to be developed, and the flexibility this system is supposed to exhibit. Assessing the four methodologies according to this evaluation framework resulted in the choice of MaSE.

3.2. Limitations of MaSE for the Embedded Systems Domain

In spite of its good support for agent-oriented concepts like goals and roles, MaSE fails to capture some of the essential characteristics of the embedded real-time systems domain. By examining the applicability of MaSE to the flexible embedded systems, a number of limitations have been identified.

3.2.1. Requirements Engineering

The lack of support to the requirements engineering phase may not have a noticeable impact on the modelling of traditional information systems whose development is based on user requirements that can be acquired based on conventional methods of software requirements engineering. However, the integrated nature of the embedded systems results in a set of constraints that stem from the technical system and from the existing system hardware. Such constraints may conflict with or limit the user requirements and need thus to be considered at early development stages. The temporal requirements of the technical system, the response time of the computational nodes, as well as the topology of the hardware components are all examples of possible factors that can greatly constrain the required system behaviour.The formal specification of these constraints is not straightforward and should be based on a careful analysis of the physical aspects of the system. Therefore, a methodology that attempts to cater for the embedded real-time application domain has to give clear support to how requirements are to be specified in light of the enforced constraints.

3.2.2. Environmental Support

In spite of the important role played by the environment in the agent-oriented paradigm, where an agent is by definition situated in an environment with which it interacts, MaSE fails to support this feature and provides no mechanism for explicitly

modelling the environment, nor for modelling the interaction between the system and its environment. Considering the embedded systems domain, the role of the environment becomes even stronger because of its integrated nature within an encapsulating device or system. Consequently, identifying the boundaries of the modelled system and designating it from its environment aids in a better understanding of the system concerned. In addition, modelling the interaction between the system and its environment is of a big significance to embedded systems due to their reactive nature, where the internal behaviour of the system is highly shaped by external events emerging from the environment.

3.2.3. Temporal Dimension of the Modelled Behaviour

While real-time requirements and constraints greatly shape the behaviour of an embedded real-time system whose performance is always judged by how far it satisfies its temporal requirements, this aspect is totally absent from the development process of MaSE and from the other methodologies that have been surveyed [7]. This is viewed as the greatest obstacle hindering the application of agent-oriented methodologies to the embedded systems domain. Hence all aspects of the system behaviour including internal behaviour of agents as well as inter-agent communications have to explicitly consider the temporal factor as a central shaping factor in the analysis and design phases.

The concurrent behaviour of the system is another aspect which is closely related to timeliness since it deals with the way the system works on satisfying several temporal requirements simultaneously. MaSE provides limited support by the concurrent tasks model generated during the analysis phase (see Fig. 1). It is assumed that each role fulfils its goals through the concurrent execution of a number of tasks whose execution details are modelled during this step. Each task is then modelled with a finite state automaton. However, the concurrency involved in managing the collective execution of these parallel tasks is not explicitly supported.

4. Proposed Extensions

In order to deal with the limitations of MaSE in conceiving flexibility to embedded systems, the whole engineering process has been refined as illustrated in Fig. 2. [1]First, a new phase for requirements engineering has been introduced. Second, modifications have been suggested to the already existing analysis and design phases.

4.1. Requirements Engineering Phase

Requirements serve in the identification of the qualitative along with the quantitative characteristics of the system [13]. They are usually viewed from two levels of abstraction. At a higher level of abstraction, requirements are described from the *user*

[1] New and modified artefacts are differentiated from conventional ones by denoting them with grey background and dashed boarders respectively.

view and are referred to as the *user requirements*. This view however is refined by the *system developer* in light of the existing constraints which results in a detailed modelling of system services and constraints which is referred to as the *system requirements* [14].

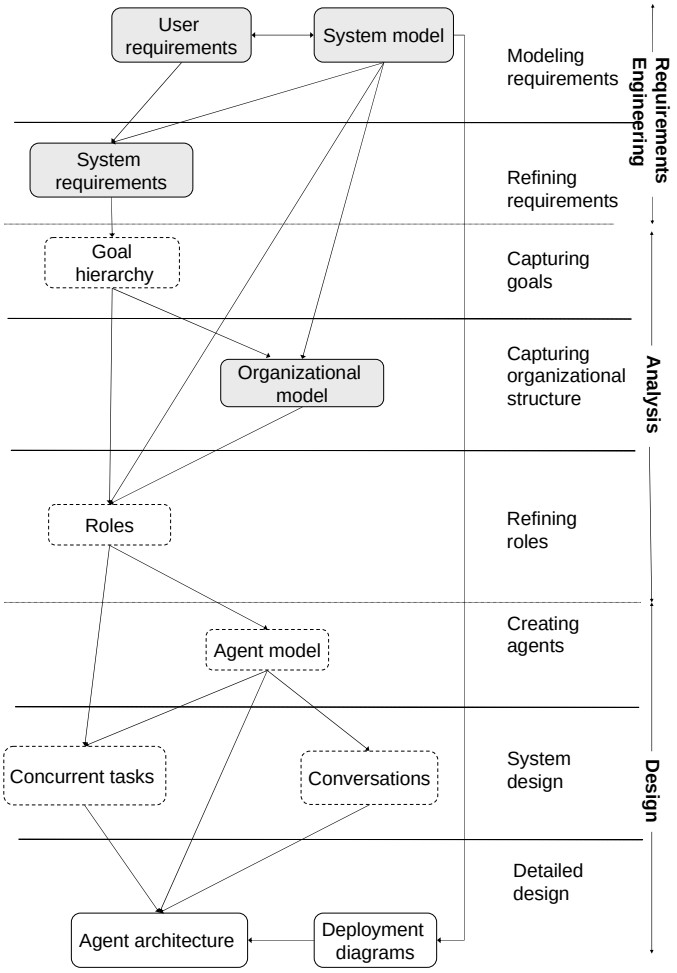

Fig. 2. The software engineering process of the extended MaSE methodology

The objective of this phase is hence to generate a refined set of systems requirements that takes the constraining effects of the technical system as well as the system hardware into consideration. This phase covers two steps that involve modelling and refinement of requirements. During the first step, requirements are modelled in the form of two artefacts: the user requirements model which incorporates user needs along with the system model which captures the system constraints, as illustrated in Fig. 2. During the modelling step, a gradual generation of

the systems requirements occurs by iterating back and forth between the modelling and the refining steps.

Basically, user requirements can be classified into process requirements and flexibility requirements. While the former relates to the basic operation of the system, the latter is associated with extra requirements that serve in exhibiting a degree of flexibility during operation. While both classifications of requirements should be analysed in light of the existing constraints, flexibility requirements are subject to feasibility analysis that might result in the elimination or modification of some of these requirements that turn out infeasible under consideration of the system constraints.

For the generation of the system requirements out of the user requirements and the system model, the following steps are proposed:

1. **Elicitation and analysis of process requirements**
   This step is concerned with the classification of the user requiremts into process and flexibility requirements. Process requirements are further analysed and refined to serve in the development of the initial set of the system requirements.

2. **Elicitation and analysis of systems constraints**
   The goal of this step is to extract the system constraints which results in the generation of the system model. This involves physical as well as behavioural analysis of the system concerned. While the former considers the static characteristics of the technical as well as the automation system, the latter attempts to study the expected behaviour of the system based on the process requirements in order to extract the relevant constraints.

3. **Analysis of the flexibility requirements**
   At the end, the flexibility requirements which resulted from the classification of the first step undergo a feasibility test based on the generated system model with the corresponding constraints. Since different aspects of flexibility may be catered for, the system modeller is advised to focus on the required aspects of flexibility. For this purpose, this step aims at establishing a view of the required flexibility. Two modelling artefacts of the SysML [15] notation are to be used for this purpose: view and viewpoint diagrams. While a view captures a certain perspective of the system, a viewpoint embodies the rules for developing a certain view. The refined flexibility requirements serve in complementing the system requirements.

The system requirements are modelled by a requirements diagram based on the SysML notation. One of the advantages of this notation is the support for associating the identified requirements with the corresponding constraints.

4.2. Environmental Support

During the requirements engineering phase, a systematic analysis of the physical structure of the automation system along with the expected behaviour of the system is carried out to extract the enforced constraints. For the sake of this analysis, the system

boundaries are identified in the form of a context diagram. By defining the boundaries of the system, a distinction is made between the system and the external environment represented in the form of external terminators that may be affected by or have an effect on the analysed system. These terminators could symbolise external systems, input/output devices, or people. A decision should be made during this stage on whether to model sensors and actuators as part of the system or as external terminators.

Interactions between the system and its external environment are captured in the form of finite state machines (FSMs). The reactive nature of the system is modelled by analysing external events and how they affect the internal state of the system. This analysis of events is performed under consideration of the temporal characteristics of these events and whether they are periodic or sporadic. The resulting FSMs are complemented with a set of events descriptors.

4.3. Adding Timeliness and Concurrency Support

In this section, the extensions that serve in overcoming the drawbacks of MaSE with respect to the support of timeliness and concurrency are highlighted.

4.3.1. Extending Goals

A goal in MaSE represents a "system-level objective" which is formulated in a way that reflects what the system is trying to achieve [16]. Analysing the system from the point of view of "what" the system is trying to achieve fails to explicitly capture the essence of the embedded real-time systems whose correctness depends not only on fulfilling the required goals but also on the timeliness of that fulfillment. It follows that goals of real-time systems have to be specified in a two-fold formulation: *what* is being aimed at, and *when* it is supposed to be achieved. In other words, while identifying goals, it is important to reason about the existence of possible deadlines for these goals. A deadline can either be absolute or relative; periodic or aperiodic; at a specified point in time or during an interval. In addition, a goal may reflect a hard or a soft real-time requirement. For example, one of the goals of a fire alarm system could be the activation of alarm in no more than time t. This reflects a hard requirement which, when violated, could lead to dangerous consequences like the spread of fire. It is worth noting that though not all goals can be assigned temporal parameters, it is recommended to examine possible temporal requirements or constraints and write them explicitly while specifying goals.

4.3.2. Extending Roles

Roles in MaSE are defined by an abstract model that associates roles with the corresponding goals which they are supposed to achieve. However, this model fails to capture the internal characteristics of roles that help in achieving the assigned goals. Therefore, the role model of the Gaia methodology [17] was adopted and extended to represent these characteristics. Roles in Gaia are defined in terms of four attributes: permissions, activities, protocols, and responsibilities. First, permissions are access rights of this role to software or hardware resources. Second, activities and protocols

represent functionalities of this role. While activities can be carried out internal to the role, protocols describe the interaction of this role and other roles. Finally, responsibilities are categorised into liveness and safety properties describing the expected behaviour that an agent playing that role should bring about and the undesired behaviour which should be avoided respectively. Under this field, the temporal constraints associated with the goals assigned to the role of concern are formulated in the form of temporal logic.

Next, the way a role achieves its goals is analysed by deciding on one of the set of alternative actions. This is made possible by first establishing a list of the role's activities that were identified in the role schema along with any relevant temporal parameters in the form of a tentative time table. Such parameters can be retrieved from the temporal constraints recorded under the responsibilities field. It is worth noting that not all activities are governed by predetermined temporal constraints and thus not all fields can be filled. However, it is worth establishing this partial list to serve in capturing the temporal requirements of the role concerned.

The dynamics of roles is then modelled by capturing how it deals with concurrent events and activities and how it decides among alternative actions. Timed Petri nets [18] were selected as a computational model that possesses the power of modelling concurrency and timeliness. Possible temporal requirements of the role recorded by its time table are to be considered and annotated in the associated Petri net.

The specification of the internal behaviour of each role is described in the concurrent tasks model in section 4.4.2.

4.3.3. Organisational Model

In complement to extending goals and roles with temporal requirements, the whole system should be realised in such a way to guarantee the satisfaction of these requirements at run time. Enforcing temporal requirements at run-time is considered starting from the analysis phase by viewing the system as an organisation of agents similar to human organisations where the freedom of members in selecting their actions is controlled by the policies and rules of their organisation. This organisational view has been proposed by other agent-oriented methodologies like Gaia [19] and Message [20]. From our point of view, a multi-agent system is conceived as a set of groups sharing a set of goals that they strive to achieve. The system as a whole as well as the individual groups are constrained by policies and rules. In order to allow for the satisfaction of these constraints, additional coordination roles may be identified. The multi-agent organisation is modelled during the analysis phase through the identification of groups, policies and roles which results in the generation of the organisational model (see Fig. 2). In addition, interaction patterns among organisational members are modelled by timed Petri-nets and incorporated into the organisational model.

4.4. Process-Related Extensions

In addition to the aforementioned extensions which were motivated by the need to tailor MaSE for the embedded real-time systems domain, a number of slight

modifications to the process were necessary for the sake of consistency and convenience.

4.4.1. Integration of the System Model

The proposed system model which captures the constraints enforced by the underlying hardware and technical system was integrated in the engineering process. As illustrated in Fig. 2, the generation of several artefacts is based either directly or indirectly on the system model. Referring to this model during the analysis and design of the system is crucial due to the constrained nature of embedded systems.

4.4.2. The Concurrent Tasks Model

This model captures the details of the internal tasks of each role in the form of finite state machines. Traditionally, this model is generated in MaSE during the analysis phase based on the role model. The system analysis should be abstracted away from such deep details which can lead to immature design decisions. Consequently, in the proposed extended methodology, this generation of this model is shifted to the design phase.

To further support the specification of temporal requirements, the finite state machines are replaced with timed automata [21]. Adopting timed automata at this stage allows for keeping compatibility with the traditional notation of finite state machines, while at the same time giving the possibility to specify all relevant temporal requirements. These temporal requirements for internal tasks of each role are directly derived from the refined role model as described in section 4.3.2.

4.4.3. Detailed Design

The support of MaSE extends up to the system deployment by capturing the distribution of agents along the available platforms in the form of a UML-based deployment diagram. This step is complemented in the extended methodology by accompanying it with the generation of the agent architecture model to form the detailed design step. The role of the agent architecture diagram in MaSE is the identification of the internal architecture of agent classes. This is either done by defining components from scratch or by the reuse of existing architecture templates [1]. With respect to the embedded systems domain with its various computational platforms and limited resources, such a decision is greatly affected by the deployment platform. Therefore, in the extended MaSE methodology, the agent architecture is generated based on the deployment diagram to design the internal architecture of agents under consideration of the characteristics of the computational platform they are going to be deployed on.

5. Evaluation of the Extended MaSE Methodology

To assess its applicability to embedded systems, the extended MaSE methodology has been evaluated based on an elevator system model. This model consists of two shafts each of which is controlled by a microcontroller and consists of four floors. The two

microcontrollers are interconnected to each other and to the peripherals by means of a CAN bus. Traditionally, each microcontroller is responsible for the separate control of one of the corresponding shaft. However, the development of a flexible control is highly desirable to improve the availability and robustness of the system by detecting and handling failures of system elements that might occur during run-time.

The development of agent-oriented software for the flexible control of the elevator system has been guided by the extended MaSE. A comprehensive coverage of the modelling process is beyond the scope of this work. The evaluation of the proposed concepts is rather illustrated by demonstrating two artefacts. First, the flexibility view is focused on to illustrate the newly introduced requirements engineering phase and how it serves in the early analysis of the flexibility requirements. Second, light is shed on the extended goal-hierarchy due to its central effect on the remaining artefacts as well as its significance in capturing the temporal constraints.

Fig. 3 captures the fault tolerance view developed during the requirements engineering phase. As illustrated in the figure, analysing the requirement of tolerating a failure of one of the microcontrollers results in the generation of three further derived requirements. These derived requirements serve in elaborating on and possibly in limiting the original requirement. Relevant constraints that govern the realisation of these requirements are identified and recorded.

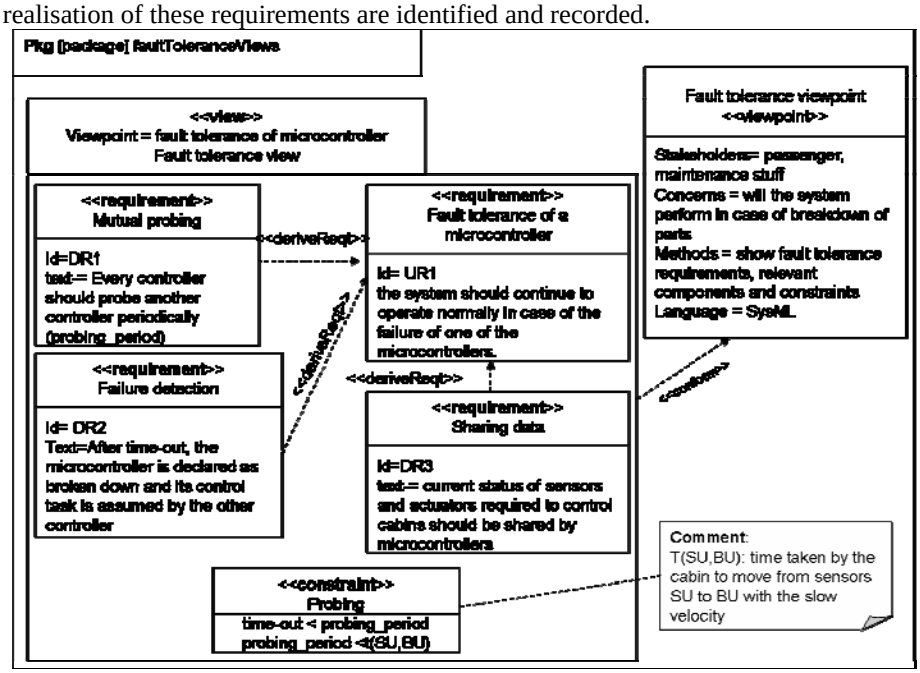

Fig. 3. Illustration of the view / viewpoint modeling of the fault tolerance aspect of flexibility

The goal hierarchy of the system is depicted in Fig. 3. Constraints that were extracted during the requirements engineering phase and incorporated in the requirements diagram are propagated to the corresponding goals in the goal hierarchy. Goal 1.1.1 is an example of a time-constrained goal which states that a passenger request has to be

acknowledged within 500 ms. The distinction between hard and soft real-time requirements is visualised by colouring each of them with a different colour.

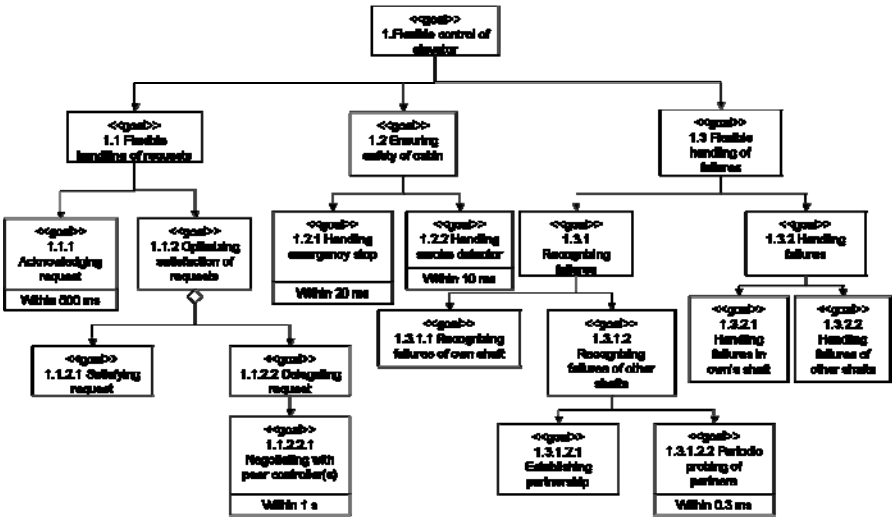

Fig. 4. The goal hierarchy of the elevator control system

6. Conclusion and Future Work

The development of flexible embedded systems – which tailor their behaviour to their dynamic environment while meeting their strict temporal requirements – is gaining an increasing attention from academia and industry. One possibility of realising flexibility is through the employment of autonomous software agents which have shown proved potential in exhibiting flexibility in the information technology field. However, the application of agents in the embedded systems domain has been hindered by the lack of concepts and methodologies that equip agents with real-time capabilities that facilitate the development of embedded real-time systems and allow for fulfilling temporal requirements at run time. This limitation of the agent-oriented software engineering has motivated this research whose objective was to extend an agent-oriented methodology for the embedded real-time systems domain.

This work is an extension to a previous study which resulted in the selection of the MaSE methodology for showing the best relative potential for the embedded systems domain based on a criteria-based evaluation specially tailored to the embedded systems domain [7]. During the course of this work, weaknesses of MaSE with respect to the development of flexible embedded real-time systems were identified and analysed. Basically, MaSE was found to suffer from a lack of support to the requirements engineering, the environmental modelling, and real-time specifications. These weaknesses have been tackled by the introduction of a requirements engineering phase which captures the timeliness constraints enforced by the underlying technical system and system hardware Environmental modelling is supported as well during the requirements engineering phase through the

identification of the boundaries of the system as a step in analysing its physical characteristics. In addition, timeliness support was proposed by extending goals and roles with real-time specifications. Finally, process-related modifications have been applied to MaSE to allow for the integration of the proposed concepts.
The extended methodology has been applied to the development of an agent-oriented flexible control of an elevator system model. Further application examples from the embedded systems domain are being currently worked on, such as the control for an industrial continuous wood press. Results of this practical evaluation will be used to further improve and refine the extended MaSE methodology.

Acknowledgement

This work has been carried out in the scope of the project AVE [19] on agent-oriented real-time systems. The project AVE is kindly funded by the German Research Council (DFG, Deutsche Forschungsgemeinschaft) under GO 810/15-1 and VO 937/5-1.

References

1. Wood, M. F.: Multiagent systems engineering: A methodology for analysis and design of multiagent systems. Master's thesis, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB Ohio, USA, (2000)
2. Lauber, R., Göhner, P.: Prozessautomatisierung 1, 1st Edition.. Springer-Verlag. (1999)
3. Kopetz, H.: Real-Time Systems Design Principles for Distributed Embedded Applications. Kluwer Academic Publishers, (1997)
4. Voros, N. S. et. al.: Hardware/Software Co-Design of Complex Embedded Systems. An Approach Using Efficient Process Models, Multiple Formalism Specification and Validation via Co-Simulation. Design Automation for Embedded Systems, 8, 5-49, (2003)
5. Gajski, D. et. al.: Specification and Design of Embedded Systems. P.T.R. Prentice Hall, (1994)
6. Parunak, H. V. D.: Practical and industrial applications of agent-based systems, Environmental Research Institute of Michigan (ERIM), (1998)
7. Mubarak, H., Göhner, P., Wannagat, A., Vogel-Heuser, B.: Evaluation of agent oriented methodologies for the development of flexible embedded real-time systems in automation. atp international, issue 1/2007, Oldenbourg Industrieverlag, München (2007)
8. Henderson-Sellers, B., Giorgini, P.: Agent-Oriented Methodologies, Idea Group Publishing, Hershey (2005)
9. Zambonelli, F, Jennings, N. and Wooldridge, M. Developing Multiagent Systems: The Gaia Methodology ACM Transactions on Software Engineering and Methodology, Vol. 12, No. 3, July 2003, Pages 317–370.

10. DeLoach, S. A.: Analysis and Design using MaSE and agentTool. In 12th Midwest Artificial Intelligence and Cognitive Science Conference (MAICS), 2001
11. Padgham, L.; Winikoff, M.: *The Prometheus Methodology*. Chapter 11 in "Methodologies and Software Engineering for Agent Systems. The Agent-Oriented Software Engineering handbook." Edited by Federico Bergenti, Marie-Pierre Gleizes and Franco Zambonelli. Kluwer Publishing, 1-4020-8057-3, July 2004. p. 217-234
12. Cossentino, M., Potts, M. A CASE tool supported methodology for the design of multi-agent systems in Proc.2002). In Proceedings of the 2002 International Conference on Software Engineering Research and Practice (SERP'02) Las Vegas, USA, June.
13. Balzert, H.: Lehrbuch der Software-Technik. Band 1. 2. Auflage. Elsevier-Verlag, (2001)
14. Sommerville, I.: Software Engineering. 6th Edition. Addison Wesley (2001)
15. OMG SysML Specification. http://xml.coverpages.org/OMG-SysML-Specification060504.pdf
16. DeLoach, S. A., Wood, M.: Multiagent Systems Engineering: the Analysis Phase. Technical Report, Air Force Institute of Technology, AFIT/EN-TR-00-02, (2000)
17. Wooldridge, M., Jennings, N., Kinny, D.: The Gaia Methodology for Agent-Oriented Analysis and Design. Autonomous Agents and Multi-Agent Systems, 3, 285.312, (2000)
18. David; R. and Alla, H. Discrete, continuous, and hybrid Petri nets. Berlin ; Heidelberg ; New York : Springer (2005)
19. Zambonelli, F., Jennings, N., Wooldridge, M.: Multi-Agent Systems as Computational Organizations: The Gaia Methodology. In Agent-Oriented Methodologies, edited by B. Henderson-Sellers and P. Giorgini, Idea Group, (2005)
20. Garijo, F. et. al.: The MESSAGE Methodology for Agent-Oriented Analysis and Design. In Agent-Oriented Methodologies, edited by B. Henderson-Sellers and P.Giorgini, Idea Group, (2005)
21. Carlson, J.: Languages and methods for specifying real-time systems, MRTC report, Mälardalen Real-Time Research Centre, Mälardalen University, (2002)
22. AVE - Agenten für flexible und verlässliche eingebettete Echtzeitsysteme. http://www.ias.uni-stuttgart.de/forschung/projekte/ave.html, (2007)