Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

A Goal-Oriented Middleware for Dependable Self-Adaptive Systems

Gabriel Siqueira Rodrigues

Monografia apresentada como requisito parcial
para conclusão do Mestrado em Computação

Orientadora
Prof.ª Dr.ª Genaina Nunes Rodrigues

Brasília
2016

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Mestrado em Computação

Coordenadora: Prof.ª Dr.ª Alba Cristina Magalhaes Alves de Melo

Banca examinadora composta por:

Prof.ª Dr.ª Genaina Nunes Rodrigues (Orientadora) — CIC/UnB
Prof. — CIC/UnB
Prof. — CIC/UnB

CIP — Catalogação Internacional na Publicação

Rodrigues, Gabriel Siqueira.
.

A Goal-Oriented Middleware for Dependable Self-Adaptive Systems / Gabriel Siqueira Rodrigues. Brasília : UnB, 2016.
45 p. : il. ; 29,5 cm.

Tese (Mestrado) — Universidade de Brasília, Brasília, 2016.

1. dependabilidade

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

A Goal-Oriented Middleware for Dependable Self-Adaptive Systems

Gabriel Siqueira Rodrigues

Monografia apresentada como requisito parcial
para conclusão do Mestrado em Computação

Prof.ª Dr.ª Genaina Nunes Rodrigues (Orientadora)
CIC/UnB

Prof. Prof.
CIC/UnB CIC/UnB

Prof.ª Dr.ª Alba Cristina Magalhaes Alves de Melo
Coordenadora do Mestrado em Computação

Brasília, 9 de Junho de 2016

Abstract

In recent years, we see a growing availability of devices with computer capabilities. Along with this advent, comes out an opportunity to develop and deploy applications that explore those devices in dynamic environments. However, such environments are inherently characterized by uncertainty, in particular from the perspective of the system designer. To design dependable solutions for environments with a high level of uncertainty, we need models at runtime to represent the system structure, requirements as well as the system's contexts of operation in an integrated way. In addition, we need methods to reason about the system levels of operation and change them at runtime, whenever needed. In this work, we propose to address those issues by devising a component-based model approach for self-adaptation relying on GORE (goal-oriented requirements engineering) and a component-based architecture model. By these means, we plan to develop a middleware that follows and implements that model. Last, but not least, the middleware we will provide fault tolerance strategies to build a foundation for dependable systems.

To design dependable solutions to environments with a high level of uncertainty we need models to represent system structure, requirements, context in an integrated way. We need also methods for reason about the system method and change it.

Keywords: dependability

Contents

List of Figures

Chapter 1

Introduction

In the last decade we have seen advances in the study of autonomic computing systems as an alternative to handle the crescent complexity of systems. The complexity in systems come with the need to execute in heterogeneous platforms, to run in multiple environments and handle multiple operation contexts.

With increasing popularity of mobile computation and wireless networks we have seen the rise of interest in new domains of computation that uses the capacity of heterogeneous computer units in a given location. Example of such domains are Ubiquitous Computing [4], Internet of Things (IoT)[2], Assisted Living[16] and Opportunistic Computing[26]. In such domains, the computing environment can greatly vary from place to place. Solutions for these domains would benefit if they could plan the deployment of the system for a given computing environment at runtime.

Salehie at al. [24] define self-adaptive software as one that adjust artifacts or attributes in response to changes in the *self* and in the *context* of a software system. *Self* being "the whole body of software". And *context* being "everything in the operating environment that affects the system's properties and its behavior".

Architecture-based (or component-based) self-adaptive approaches to implement self-adaptive systems adapt the systems by acting on components of the system or by replacing them[10]. A possible solution for development of applications for environment with a high level of uncertainty would be to choose an architecture at runtime for the given environment, distributing software components between available computational units and setting up the right communication channels. But to make it possible we need also a model of the system so we can reason about possible system structures in face of the system requirements and context. So it is important to trace system components back to requirements.

Goal Oriented Requirements Engineering (GORE) approaches have gained special attention as a technique to specify self-adaptative systems[20]. Goals capture, the various objectives the system under consideration should achieve[28].

A multi-agent system (MAS) is a distributed computing system with autonomous interacting intelligent agents that coordinate their actions so as to achieve its goals[29].

Autonomous software agents provide a promising solution to the needs of decentralized networked systems, able to adapt their behaviour in a complex and dynamically changing environment [20].

Dalpiaz et al.[7] proposed Runtime Goal Models to reason about runtime fulfillment of goals.

Goal models allow us to reason about the requirements of the system and its execution context, however lacks a structural view of the system to be. On the other hand, component-based software engineering (CBSE) explicitly addresses the system structure. To the best of our knowledge there is no integrated model to reason about the system structure in relation to its contexts and requirements. In addition, we need methods to reason about the system levels of operation and change them at runtime, whenever needed. In this work, we propose to address those issues by devising a component-based model approach for self-adaptation relying on GORE (goal-oriented requirements engineering) and a component-based architecture model.

Such model would allow us to built more dependable systems. For example, we could choose to use fault tolerance techniques to more critical tasks of the system.

In this work we will explore the integration of approaches to allow reasoning about the relationship between the system goals, components and context of execution.

The objective is be able to create valid deployment configurations, trace the system goals accomplishment at runtime and use fault tolerance techniques to improve the dependability of the system in face of error prone components.

In order to allow the system make decisions about its structure based on requirements and context we need a model that can correlate these three concepts: the system structure, the goals and the execution context.

Research Question 1 (RQ1): What would be a good model of software system that could allow one to reason about the system structure, context and trace the goals at runtime? In other words, how to represent the system goals, architectural structure, operation context and their relationship?

Previous work in the literature [22] have partially tackled this research question. However, our major concern here relates to the traceability between goals and tasks, software architectural components and the context of operation.

To develop what would be a good model we address further questions in relation to the fitness of the model for the purpose of deciding on system adaptations. First of all, we want to realize a valid deployment of the system:

Research Question 2 (RQ2): How to, using the model from RQ1, deploy a Self-Adaptive System that is dependable in face of context variability?

Beside checking the system validity in other to predict system dependability, we want to be able to tolerate faults at runtime. This leads us to the next research question:

Research Question 3 (RQ3): How to guarantee that the systems is dependable in face of error prone components at runtime?

1.1 General Objectives

- propose, implement and validate a model to reason about and adapt systems based on its adaptation goals, structure and context.

1.2 Specific Objectives

- propose a conceptual model for component-based runtime goal-model system
- implement and validate a middleware for component-based runtime goal model
- release the middleware as a comprehensible open source project
- implement and validate runtime strategies for dependability in the proposed platform

Chapter 2

Background

2.1 Dependability

Dependability can be defined as the ability of a system to avoid faults in its services that (1) are more frequent or (2) more severe than acceptable. Or as the characteristic of a system to be justifiably trusted.

A common terminology used for system deviations is the following: [3]

- **failure**: (or service failure) is a perceived deviation from the correct service provided by a system.
- **error**: is a deviation of correct internal system state that can lead to its subsequent failure.
- **fault**: is the adjudged or hypothesized cause of an error

Dependability includes the following attributes:[3]

- **availability**: readiness for correct service.
- **reliability**: continuity of correct service.
- **safety**: absence of catastrophic consequences on the user(s) and the environment.
- **integrity**: absence of improper system alterations.
- **maintainability**: ability to undergo modifications and repairs.

Many means have been developed of how to attain the attributes of dependability. These means can be classified as:

- **Fault prevention** means to prevent the occurrence or introduction of faults.
- **Fault tolerance** means to avoid service failures in the presence of faults.
- **Fault removal** means to reduce the number and severity of faults.
- **Fault forecasting** means to estimate the present number, the future incidence, and the likely consequences of faults.

2.2 Attain Dependability at Runtime

To keep dependability in face of uncertainty in the deployment environment some techniques have been proposed for runtime analysis at runtime.

Felipe et al[11] propose a method of fault-tolerance for a scientific workflow execution in grid.

Alessandro Leite [8] propose a fault tolerance schema for cloud deployment based on which a fault instance in the cloud is monitored and in case of failure the instance can be restarted or terminated and them a new instance created.

Danilo et al[18] propose a methodology for fault forecasting by which developer, at design time, annotate the goal decomposition in goal model and specify context variables. A special tool generate a formula for, given a context, evaluate the probability of achieve a goal at runtime.

2.3 Self-Adaptive Systems

Self-adaptive systems have been accepted as a promising approach to tackle context change. Self-adaptivesses is an approach in which the system *"evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible."*[17].

Self-adaptive software aims to adjust various artifacts or attributes in response to changes in the self and in the context of a software system[24].

A key concept in self-adaptive systems is the awareness of the system. It has two aspects[24]:

- *self-awareness* means a system is aware of its own states and behaviors.
- *context-awareness* means that the system is aware of its context,

Schilit et al.[15] define *context* as "the sufficiently exact characterization of the situations of a system by means of perceivable information that is relevant for the adaptation of the system".

Schilit et al.[15] define *context adaptation* as "a system's capability of gathering information about the domain it shares an interface with, evaluating this information and changing its observable behavior according to the current situation".

2.4 Software Components and Architecture

Heineman define *software component* as a "software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard"[13].

Software components is a unit of composition. Software systems are build by composing different components. Software components must conform to a component model by having contractually specified interfaces and explicit context dependencies only.[27].

A *component interface* "defines a set of component functional properties, that is, a set of actions that's understood by both the interface provider (the component) and user

(other components, or other software that interacts with the provider)"[6]. A component interface has a role as a component specification and also a means for interaction between the component and its environment. A *component model* is a set of standards for a component implementation. These standards can standardize naming, interoperability, customization, composition, evolution and deployment.[13]

The *component deployment* is the process that enables component integration into the system. A deployed component is registered in the system and ready to provide services[6].

Component binding is the process that connects different components through their interfaces and interaction channels.

Software architecture deals with the definition of components, their external behavior, and how they interact.[14]

Component based software engineering (CBSE) approach consists in building systems from components as reusable units and keeping component development separate from system development[6].

CBSE is built on the following four principles[6]:

- Reusability. Components, developed once, have the potential for reuse many times in different applications.
- Substitutability. Systems maintain correctness even when one component replaces another.
- Extensibility. Extensibility aims to support evolution by adding new components or evolving existing ones to extend the system's functionality.
- Composability. System should supports the composition of functional properties (component binding). Composition of extra functional properties, for example composition of components' reliability, is another possible form of composition.

2.5 Goal-oriented requirements engineering

Goal-oriented requirements engineering (GORE) is concerned with the use of goals for eliciting, elaborating, structuring, specifying, analyzing, negotiating, documenting, and modifying requirements[28].

GORE models are the main tool used by system analysts and stakeholders to reason about the system requirements. Goal modeling represents a shift in relation to traditional software development approaches as it focus on stakeholder goals and states that the system needs to achieve and not in how it achieves it[1]. Goal models are graphs representing AND/OR-decomposition of abstract goals down to operationalisable leaf-level goals. [21]

A goal is an objective the system under consideration should achieve. [28]

2.6 TROPOS

Tropos[5] is a methodology for develop multi-agent systems that uses goal models for requirement analyses. Tropos encompasses the software development phases, from Early Requirements to Implementation and Testing.

2.6.1 The Tropos key concepts

The methodology adopts the i* [30] modeling framework, which proposes the concepts of actor, goal, task, resource and social dependency to model both the system-to-be and its organizational operating environment[5]. In more recent publication [19] about the Tropos modeling framework the concept of *task* was renamed to *plan*.

The following are the key concepts in the Tropos metamodel[19]:

- Actor: an entity that has strategic goals and intentionality
- Goals: it represents actors' strategic interests. Hard goals are goals that have clear-cut criteria for deciding whether they are satisfied or not. *Softgoals* have no clear-cut criteria and are normally used to describe preferences and quality-of-service demands.
- Plan: it represents, at an abstract level, a way of doing something. The execution of a plan can be a means for satisfying a goal or for *satisficing* (i.e. sufficiently satisfying) a softgoal.
- Resource: it represents a physical or an informational entity.
- Dependency: its a relationship between to actors that specify that one actor (the depended) have a dependency to other actor (the *dependee*) to attain some goal, execute some plan or deliver a resource. The object of the dependence is the dependum.
- Capability: it represents both the *ability* of an actor to perform some action and the *opportunity* of doing this.

2.7 Multiagent systems (MAS)

According to Wooldridge[29], Multiagent systems (MAS) are systems composed of multiple interacting computing elements known as agents. Agents are computer systems that are capable of autonomous action and interacting with other agents.

Chapter 3

Proposal

3.1 Proposed Solution

We propose a component model for goal oriented multi-agent systems and a runtime model that allows reason about the system levels of operation.

This model will be implemented by a middleware, that will keep the traceability between goals and architecture and allow for managing the architecture and implementing fault-tolerance in face of fault components at runtime.

3.2 Conceptual Model

3.2.1 Component Model

We propose and extension to Tropos Model with a component model. By this we aim at creating an appropriate abstraction to allow composable architecture while keeping the traceability between the requirements and implementation at runtime.

Or component model is build around the concept of *strategy*. The strategy at goal model level is a mean of achieving a particular goal. It can have the following realization at runtime:

- a strategy can be implemented by a component in the architecture.
- can be a delegation to another known agent as a runtime decision.
- can be a fault tolerant proxy that combines a implementation or delegation with a fault tolerance technique.

From a goal model, each goal will originate at least a capacity and strategy. Each OR decomposition of a goal, in the goal model, should correspond to an additional strategy. An agent in the system has a repository of strategies.

The main concepts are:

- **Agent**: agent is an independent computational unit that manages its own resources (CPU, memory, disk, sensors, etc). After system deployment an agent is turned into an actor of a Tropos model.

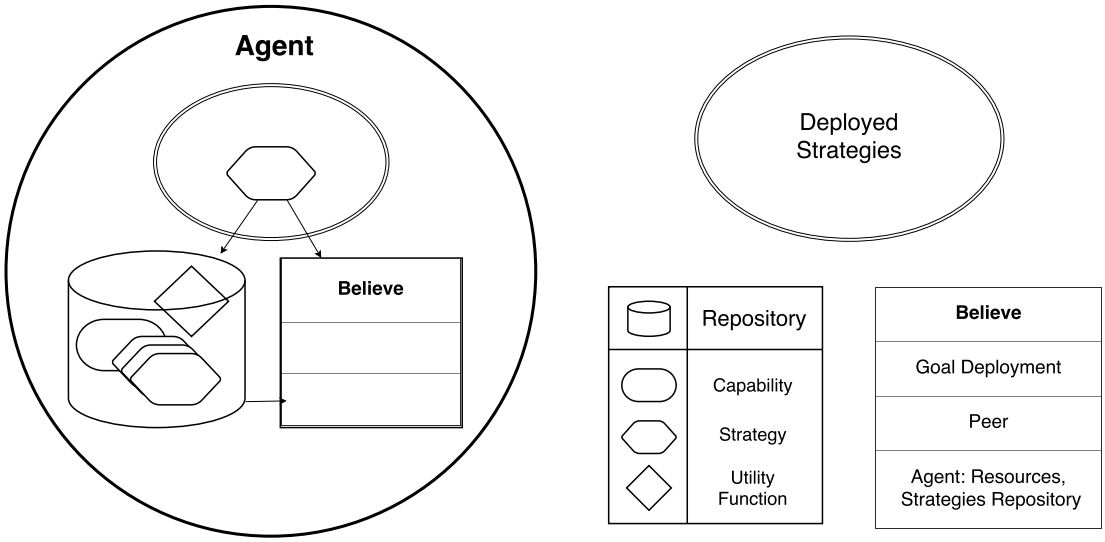

Figure 3.1: Agent Respresentation

- **Capability**: description of a kind of goal that an agent can perform. Its a component interface description in the architecture. (e.g SUM a and b).
- **Strategy**: a strategy is a capability alternative implementation (e.g (SUM,a,b) => a+b). Its also a module in the architecture. A strategy should implement one capability.
- **Strategy Repository**: an agent has a repository of its capabilities and strategies.
- **System Deployment**: the process of putting the right strategies in agents repositories.

3.2.2 Architectural Layers

For the implementation of the conceptual model, we propose an architecture of three layers:

- **Infrastructure**: This layer is responsible for the component model implementation, how capability, strategies, plans and goals instances are described. It also responsible for agent and goals life cycle, strategies management and selection.
- **Adaptation**: This layer is responsible for keep the level of service. It contains strategies to discover peers, team up with peers, gather information to improve strategy selection, etc.
- **Application**: This layer if responsible for functional strategies. The functional strategies are the one that implement user application.

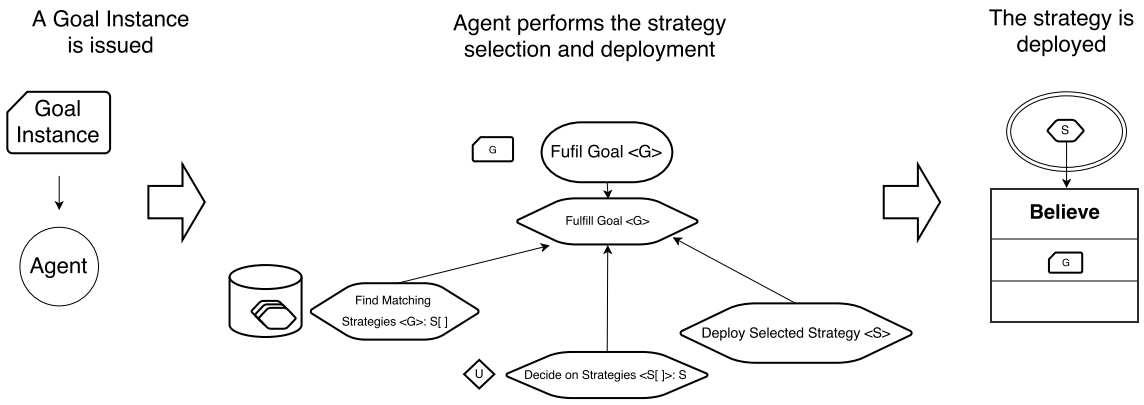

Figure 3.2: The deployment of a strategy

3.2.3 Runtime Model

Dalpiaz et al. [7] argue that traditional goal models are not enough for reason about the system at runtime. They propose a distinction between Desgin-time Goal Model (DGM) and Runtime Goal Model (RGM). We extend their proposal of RGM with a component model and runtime strategy selection.

- **Goal Instance**: an actual instance of an objective for a given data set. (e.g SUM 2 and 3)
- **Believe**: in the model of an actor about itself and the context.
- **Strategy Deployment**: occurs after the strategy selection, is the the binding between the strategy, its dependencies and its environment dependencies.

In the proposed model an actor achieve a goal by *deploying a strategy*. For instance the deployment of a strategy is a capability itself.

3.2.4 Strategy Deployment

In order to allow component based adaptation we propose a mechanist of strategy selection at runtime. This mechanism is part of the infrastructure layer. For a hight level of flexibility we propose that the selection mechanism itself should be implemented by components in the architecture.

At a low level an agent has the capability of fulfill goals. Inspired by component based frameworks like Rainbow[9] we propose that the strategy should be chosen by means of a utility function. That utility function should be responsible to calculate which available strategy will have a better contribution for softgoals.

After a strategy is selected it is deployed. Strategy deployment corresponds to a component binding in CBSE.

The capacity <fulfill goal> and its corresponding strategy should be as follows:

- <**Fulfill Goals**> the capability to fulfill generic goals.

- <**Fulfill Goals**> strategy to fulfill goals by selecting available strategies and evaluating them with an utility function. Consists of 3 sub-goals:
  - Find Matching Strategies
  - Decide on Strategies
  - Deploy the Selected Strategy

And the following three strategies implement the previous 3 capabilities.

- <**Find Local Matching Strategies**> accomplishes <Find Matching Strategies> returns the list of matching strategies. A matching strategy is any strategy that implements the goal interface.
- <**Select a Strategy**> accomplishes <Decide on Strategies> using a pre-configured utility function that analyses strategies metadata and select a strategy.
- <**Deploy Strategy**> accomplishes <Deploy the Selected Strategy>. Bind the components with the goal instance an dependent strategies.

3.2.5 Awareness

The agent have a model of that repository that it can use for reason about its capacities, strategies and plans. The agent is also able to manage its own repository.

Self-awareness is provided by a set of self-awareness strategies. Example of self-aware strategies are:

- <**List Local Strategies**> **Strategy** How the agent can know it actual capabilities.
- <**List Goal Instances**> **Strategy** How the agent can know it current intentions.
- <**List Deployed Strategies**> **Strategy** How the agent can know it current running strategies.

3.3 Related Work

Rainbow is a framework for self-adaptation architecture based[9]. It keeps an model of the architecture of the system and can be extended with rules to analysis the system behavior at runtime, find adaptation strategies and perform this changes. It separate the functional code (internal mechanisms) from adaptation code (external mechanism) in a schema called external control, influenced by control theory. [10] Different from our proposal it isn't goal-oriented an there is no work on how to relate Rainbow components to requirements.

MUSIC project provides a component-based middleware for adaptation that propose to separate the self-adaptation from business logic and delegate adaptation logic to generic middleware. As in our propose it adapts by evaluating in runtime the utility of alternatives, to chose a feasible one (e.g., the one evaluated as with highest utility)[23]. As Rainbow, MUSIC is not goal-oriented.

Salehie et al. [25] propose a run-time goal model and its related action selection. It models adaptable software as a system that exposes sensors and effectors and proposes a model consisting in Goals, Attributes and Action for selecting actions that will effect the adaptable software at runtime, giving sensed attributes. So the adaptation mechanism is to choose the best action given the actual attributes. As this work it uses explicit runtime goals and make them visible and traceable. Different from it we use a more symmetric approach that can allow for functional and adaptation management.

Günalp et al. [12] propose a middleware for pervasive software with autonomic capabilities. The approach is service based. It proposes a component written in a custom language and the use of components repository that allows the discovery on new sensors. The system present a support for adaptability by using policies.

Chapter 4

Methodology

In this section we will describe our methodology. At a hight level our main activities will be: review the literature, elaborate a conceptual model, implement and evaluate a middleware, write the dissertation, write paper and viva presentation. The implementation will be divided in 2 phases so we can submit intermediate results to scientific conferences.

1. *literature review*

   First we will conduct a literature review with the objective to find all the relevant related works, what is the state of the art, review that works and better proposition our proposal in relation to that works.

2. *elaborate a conceptual model and architecture*

   In parallel to the literature review, we will develop a conceptual model to integrate Goal-Oriented Requirements Engineering, Architecture-Based Self-Adaptation so that the system goals can be traced to architecture modules at runtime. This conceptual-model will describe the Architectural components of the proposed architecture, their relations, how this elements are related to the goal model that they implement and how this model can be manipulated at runtime.

3. *implement and validate a middleware (Core)*

   At this stage we will implement the core of a middleware to support execution of components as described by the proposed conceptual model and evaluate. To do the evaluation we will specify and describe a study case system and implement the system using the middleware and the proposed architecture.

4. *write paper about the middleware (Core)*

   We will report the results of the middleware implementation and validation as a paper.

5. *implement and validate dependability strategies*

   implement and evaluate GODA[18] strategy for runtime dependability analysis at runtime.

6. *write paper about dependability strategies*

We will report the results of the middleware implementation and validation as a paper.

7. *write up dissertation*

   gather all intermediate results and format as a dissertation.

8. *viva*

   present the dissertation.

Chapter 5

Expected Results

5.1 Expected Results

With this work we expect to:

- Collaborate to self-adaptation corpus of knowledge by contributing with a conceptual model that maps concepts of Goal-Model and Architecture-based adaptation.
- Generate a proposal of architecture and reference middleware for development of applications that face a high level of context variation and couple this uncertainty by reasoning about its goal model at runtime.
- By the former two items, we expect to allow for future development of more flexible and dependable and adaptable software system.

Chapter 6

Chronogram

The proposed work is composed of the following activities:

1. *literature review*
2. *elaborate a conceptual model*
3. *implement and validate a middleware (Core)*
4. *write paper about the middleware (Core)*
5. *implement and validate dependability strategies*
6. *write paper about dependability strategies*
7. *write up dissertation*
8. *viva*

Activity	prior Aug/15	Sep/15	Out/15	Nov/15	Dez/15	Jan/16	Fev/16	Mar/16	Apr/16	May/16	Jun/16
1	X	X									
2		X	X								
3			X	X	X						
4			X	X	X						
5						X	X	X			
6						X	X	X			
7									X	X	X
8											X

Table 6.1: Schedule of Proposed Activities

Referências

[1] Raian Ali, Fabiano Dalpiaz, and Paolo Giorgini. A goal-based framework for contextual requirements modeling and analysis. *Requirements Engineering*, 15(4):439–458, July 2010. 6

[2] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A Survey. *Comput. Netw.*, 54(15):2787–2805, October 2010. 1

[3] Algirdas Avižienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):11–33, 2004. 4

[4] Genevieve Bell and Paul Dourish. Yesterday's Tomorrows: Notes on Ubiquitous Computing's Dominant Vision. *Personal Ubiquitous Comput.*, 11(2):133–143, January 2007. 1

[5] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, May 2004. 6, 7

[6] Ivica Crnkovic, Judith Stafford, and Clemens Szyperski. Software Components beyond Programming: From Routines to Services. *IEEE Software*, 28(3):22–26, 2011. 6

[7] F. Dalpiaz, A. Borgida, J. Horkoff, and J. Mylopoulos. Runtime goal models: Keynote. In *2013 IEEE Seventh International Conference on Research Challenges in Information Science (RCIS)*, pages 1–11, May 2013. 2, 10

[8] Alessandro Ferreira Leite. *A user centered and autonomic multi-cloud architecture for high performance computing applications.* PhD thesis, Paris 11, 2014. 5

[9] D. Garlan, Shang-Wen Cheng, An-Cheng Huang, B. Schmerl, and P. Steenkiste. Rainbow: architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54, October 2004. 10, 11

[10] David Garlan, Bradley Schmerl, and Shang-Wen Cheng. Software Architecture-Based Self-Adaptation. In Yan Zhang, Laurence Tianruo Yang, and Mieso K. Denko, editors, *Autonomic Computing and Networking*, pages 31–55. Springer US, 2009. 1, 11

[11] Felipe Pontes Guimaraes, Pedro Célestin, Daniel Macedo Batista, Genaína Nunes Rodrigues, and Alba Cristina Magalhaes Alves de Melo. A Framework for Adaptive Fault-Tolerant Execution of Workflows in the Grid: Empirical and Theoretical Analysis. *Journal of Grid Computing*, 12(1):127–151, October 2013. 5

[12] Ozan Günalp, Levent Gürgen, Vincent Lestideau, and Philippe Lalanda. Autonomic Pervasive Applications Driven by Abstract Specifications. In *Proceedings of the 2012 International Workshop on Self-aware Internet of Things*, Self-IoT '12, pages 19–24, New York, NY, USA, 2012. ACM. 12

[13] George T. Heineman and William T. Councill, editors. *Component-based Software Engineering: Putting the Pieces Together*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001. 5, 6

[14] Arvinder Kaur and Kulvinder Singh Mann. Component Based Software Engineering. *International Journal of Computer Applications*, 2(1):105–108, May 2010. Published By Foundation of Computer Science. 6

[15] Cornel Klein, Reiner Schmid, Christian Leuxner, Wassiou Sitou, and Bernd Spanfelner. A Survey of Context Adaptation in Autonomic Computing. pages 106–111. IEEE, March 2008. 5

[16] Thomas Kleinberger, Martin Becker, Eric Ras, Andreas Holzinger, and Paul Müller. Ambient Intelligence in Assisted Living: Enable Elderly People to Handle Future Interfaces. In *Proceedings of the 4th International Conference on Universal Access in Human-computer Interaction: Ambient Interaction*, UAHCI'07, pages 103–112, Berlin, Heidelberg, 2007. Springer-Verlag. 1

[17] Robbert Laddaga. Self Adaptive Software SOL BAA 98 12. Technical report, 1997. 5

[18] Danilo Mendonça. Dependability Verification for Contextual/Runtime Goal Modelling, 2015. 5, 13

[19] Mirko Morandini, Fabiano Dalpiaz, Cu Duy Nguyen, and Alberto Siena. The Tropos Software Engineering Methodology. In Massimo Cossentino, Vincent Hilaire, Ambra Molesini, and Valeria Seidita, editors, *Handbook on Agent-Oriented Design Processes*, pages 463–490. Springer Berlin Heidelberg, 2014. 7

[20] Mirko Morandini, Frédéric Migeon, Marie-Pierre Gleizes, Christine Maurel, Loris Penserini, and Anna Perini. A Goal-Oriented Approach for Modelling Self-organising MAS. In Huib Aldewereld, Virginia Dignum, and Gauthier Picard, editors, *Engineering Societies in the Agents World X*, number 5881 in Lecture Notes in Computer Science, pages 33–48. Springer Berlin Heidelberg, November 2009. 1

[21] Mirko Morandini, Loris Penserini, and Anna Perini. Operational Semantics of Goal Models in Adaptive Agents. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '09, pages 129–136, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems. 6

[22] Leonardo Pessoa, Vander Alves, Paula Fernandes, Genaína Nunes Rodrigues, Hervaldo Carvalho, and Thiago Castro. Dependable Dynamic Software Product Lines: an Application into the Body Sensor Network Domain. *Not published yet*, 2015. 2

[23] Romain Rouvoy, Paolo Barone, Yun Ding, Frank Eliassen, Svein Hallsteinsen, Jorge Lorenzo, Alessandro Mamelli, and Ulrich Scholz. MUSIC: Middleware Support for Self-Adaptation in Ubiquitous and Service-Oriented Environments. pages 164–182. Springer-Verlag, Berlin, Heidelberg, 2009. 11

[24] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive Software: Landscape and Research Challenges. *ACM Trans. Auton. Adapt. Syst.*, 4(2):14:1–14:42, May 2009. 1, 5

[25] Mazeiar Salehie and Ladan Tahvildari. Towards a Goal-driven Approach to Action Selection in Self-adaptive Software. *Softw. Pract. Exper.*, 42(2):211–233, February 2012. 12

[26] Stephen D. Smaldone. *Improving the Performance, Availability, and Security of Data Access for Opportunistic Mobile Computing.* PhD thesis, Rutgers University, New Brunswick, NJ, USA, 2011. AAI3474990. 1

[27] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002. 5

[28] Axel Van Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour. In *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering*, RE '01, pages 249–, Washington, DC, USA, 2001. IEEE Computer Society. 1, 6

[29] Michael Woolridge and Michael J. Wooldridge. *Introduction to Multiagent Systems.* John Wiley & Sons, Inc., New York, NY, USA, 2001. 1, 7

[30] Eric Siu-Kwong Yu. *Modelling Strategic Relationships for Process Reengineering.* PhD thesis, University of Toronto, Toronto, Ont., Canada, Canada, 1996. UMI Order No. GAXNN-02887 (Canadian dissertation). 7