TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

MASTERARBEIT

# Criteria-Driven Scheduling in an IEEE-FIPA Compliant Multi-Agent Infrastructure

ausgeführt am

Institut für Medizinische Kybernetik und Artificial Intelligence
der Medizinischen Universität Wien

sowie am

Österreichischen Forschungsinstitut für Artificial Intelligence

unter der Anleitung von

o.Univ.-Prof. Ing. Dr. Robert Trappl

und

Univ.-Ass. Dipl.-Ing. Dr. Paolo Petta

als verantwortlich mitwirkendem Universitätsassistenten

durch

**Christoph David Hermann**
Kautzenerstraße 7
A-3860 Heidenreichstein

Datum

Unterschrift

**Zusammenfassung**

Multiagentensysteme stellen einen möglichen Ansatz zur Lösung von verteilten Problemen dar. Im Gegensatz zu klassischen verteilten Problemlösungsstrategien können in einem Multiagentensysteme mehrere, möglicherweise konkurrierende Ziele existieren. Die *IEEE Foundation for Intelligent Physical Agents*, kurz FIPA, ist bemüht einen generellen, kommerziell nutzbaren Standard zu schaffen der klar das Verhalten einzelner Komponenten des Multiagentensystems regelt, jedoch nicht auf die konkrete Implementierung der Teilsysteme eingeht. Zusätzlich zur Systeminfrastruktur spezifiziert FIPA eine Menge von Interaktionsprotokollen zur Koordination von Agenten. FIPA-konforme Multiagentensysteme bieten jedoch keine direkte Unterstützung für Multiagentenplanung. Im Zuge dieser Masterarbeit wird ein neuer Ansatz zur FIPA-konformen Integration von Multiagentenplanung und Scheduling vorgestellt, der auf TÆMS und VIE-CDS aufbaut.

*Framework for Task Analysis, Environment Modeling and Simulation*, kurz TÆMS, bietet ein formales, domänenunabhängiges Modell zur Repräsentation qualitativer und quantitativer Aspekte von Koordination ebenso wie ein Modell zur Lösung von Multiagentenplanungsproblemen unter weichen Echtzeitbedingungen. *Vienna Criteria-Driven Scheduler*, kurz VIE-CDS, ist ein *Design-to-Criteria* Scheduler welcher auf TÆMS-Strukturen arbeitet. Sogenannte Schedulingkriterien erzwingen einen Kompromiss zwischen Qualität, Kosten und Dauer der Scheduleausführung.

Der im Rahmen dieser Masterarbeit spezifizierte Prozess einer geschedulten Aktionsausführung besteht aus fünf Phasen und nutzt wiederum nur FIPA-standardisierte Interaktionsprotokolle und Systemkomponenten. Dies garantiert die Anwendbarkeit des Ansatzes auf FIPA-konformen Plattformen.

Basierend auf der ebenfalls neu entworfenen XTAEMS Kodierung von TÆMS-Strukturen erfolgte eine exemplarische Implementierung an Hand des *Java Agent Development Framework*, kurz JADE, als Repräsentanten für eine FIPA-konforme Entwicklungsumgebung.

**Abstract**

Multi-agent systems provide a possible approach for solving distributed problems. Unlike conventional distributed problem solving strategies multi-agent systems can handle distinct and maybe competing goals in the same system. The *IEEE Foundation for Intelligent Physical Agents* referred to as FIPA defines standards for multi-agent systems by specifying system components at functional and not implementation level. Its aim is to promote commercially usable agent-based technologies. Additionally, FIPA specifies so called interaction protocols serving agent coordination. FIPA-compliant agent platforms do not directly offer multi-agent planning facilities. This master's thesis presents a new approach for integrating multi-agent planning and scheduling in a FIPA-compliant multi-agent infrastructure.

*Framework for Task Analysis, Environment Modeling and Simulation* referred to as TÆMS provides a formal domain-independent framework to represent qualitative and quantitative aspects of coordination as well as a model for solving multi-agent planning and scheduling problems under soft realtime conditions. *Vienna Criteria-Driven Scheduler* referred to as VIE-CDS is a *Design-to-Criteria* scheduler working on TÆMS structures. So called scheduling criteria allow to determine preferences for schedules concerning quality, cost and duration of schedule execution.

In this master's thesis, the process of scheduled action execution is defined by five phases using only FIPA standardised interaction protocols. This approach guarantees applicability to FIPA compliant multi-agent systems.

Based on the new XTAEMS encoding of TÆMS structures, an example implementation using the *Java Agent Development Framework* referred to as JADE, being a representative of FIPA-compliant multi-agent infrastructures, is provided.

# Table of Contents

# List of Figures

# Listings

# 1. Introduction

Multi-agent systems can be used for distributed problem solving in more or less open domains. In contrast to classical distributed problem solving, agents in multi-agent systems (MAS) need not share a single top-level goal. They have their own goals and capabilities, and the goals of different agents may well be conflicting. Furthermore, agents are able to actively engage in interactions with one another. Problem solving abilities of the MAS emerge from agent interaction and combination of their individual skills.

In this master's thesis we deal with *autonomous and collaborative agents*. Autonomy means that agents are not stuck with the information provided by their designers. They are able to adapt their knowledge according to the changing environment they live in. Collaboration addresses the fact that agents tend to offer their capabilities to others and do not fool one another deliberately for gaining advantage. To realise such problem solving capabilities, it is necessary that a community of agents acts in a coherent manner to achieve such problem solving capabilities. Hence, it follows that there must be some form of coordination between agents to put their actions in an order supporting the overall problem solving process.

Several techniques for coordination between agents have been developed and implemented in different multi-agent system frameworks. *Task Analysis, Environment Modeling, and Simulation* (TÆMS) [Lesser et al., 2004] is a formal domain-independent framework which

> *" . . . models problem-solving activities of an intelligent agent operating in environments where responses by specific deadlines may be required, where the information required for the optimal performance of a computational task may not be available, where the results of multiple agents' computations (to interdependent sub-problems) may need to be aggregated together in order to*

*solve a high-level goal, and where an agent may be contributing concurrently to the solution of multiple goals."*

[Horling et al., 1999, p.4]

Directly associated with TÆMS is *Design-to-Criteria*[1] (DTC) scheduling [Wagner et al., 1998] which computes custom tailored schedules for achieving high-level goals represented by TÆMS structures. *Java Agent DEvelopment Framework*[2] (JADE) [Bellifemine et al., 2007] is a software framework which eases development of multi-agent systems through an FIPA[3] compliant middle-ware. It offers a comprehensive communication infrastructure and various coordination mechanisms but lacks support for multi-agent planning and scheduling. Inspired by *coordination as a service* [Viroli and Omicini, 2006] we present a mechanism to integrate multi-agent planning and scheduling, namely VIE-CDS[4] [Jung, 2003], with JADE.

In the remainder of this chapter further concepts and tools concerning our integration mechanism of VIE-CDS with JADE will be introduced and discussed. Chapter 2 explains our XML-based encoding of TÆMS structures, refactoring and extension of VIE-CDS. Chapter 3 deals in detail with the integration of VIE-CDS in JADE. Chapter 4 demonstrates the key concepts of our approach in a medical care toy example. Chapter 5 summarises the key issues of this master's thesis, sketches a future perspective of criteria-driven scheduling in JADE, proposes further investigations on this topic and gives a review of related work.

## 1.1. Intelligent Agents?

There is no commonly agreed, exact definition about what an *intelligent agent* is. The first reason for that state of affairs is that the term agent is not owned by AI research. Agent is an everyday life vocable. We know human agents, such as travel agents, real estate agents or generally speaking people in terms of salespersons or representatives. The medical and technical sciences also have adopted the term agent for their purposes, as in active agent, abstergent agent, adhesive agent, etc. The second reason is that

[1] http://dis.cs.umass.edu/research/dtc/, last visited May 2, 2007.
[2] http://jade.tilab.com/, last visited May 2, 2007.
[3] http://www.fipa.org/, last visited May 2, 2007.
[4] http://www.ofai.at/research/agents/projects/viecds/, last visited May 2, 2007.

no generally accepted definition of intelligence with respect to software exists. In the following, we want to give an impression of what makes a piece of software an agent. Additionally, we replace the term intelligent by *rational*. We regard an agent as intelligent if it acts rationally.

In the simplest case, anything *perceiving* its *environment*[5] through *sensors* and acting upon that environment through *actuators* can be viewed as an agent. Agents act *rational* if they maximise their performance measure. This performance measure assesses how successfully the agent behaves in its environment [Russell and Norvig, 2003]. In terms of Michael Wooldrige and Nicholas R. Jennings [Wooldridge and Jennings, 1995] rationality is (roughly) the assumption that an agent will act so as to achieve its goals. They distinguish a weak and a stronger notion of agency. A computer system must fulfil the following properties to satisfy the *weak notion of agency*:

**Autonomy:** Agents work without human intervention and have control over their internal state and goals. This means that control of when to start action execution lies with the executing agent; in contrast to method invocation in object oriented languages, where the control of start lies with the caller object.

**Social ability:** Agents interact with other agents via some *agent communication language*[6].

**Reactivity:** As mentioned above, agents perceive their environment. Reactivity refers to the fact that agents respond to these perceptions according to their needs and goals.

**Pro-activeness:** Pro-activeness complements reactivity. Agents do not simply act in response to changes in the environment. Instead, they are able to take the initiative while heading for their goals.

If we concede some more human concepts like *emotions* or *mentalistic* notions like beliefs, desires and intentions to agents, we follow the *stronger notion of agency*. Indeed, intentional notions may serve as an abstraction tool to describe, explain and predict complex systems without requiring more detailed information about their internal design [Wooldridge and Jennings, 1995].

[5]For a discussion on environments please refer to section 1.1.2.

[6]For details on agent communication languages please refer to section 1.1.4.

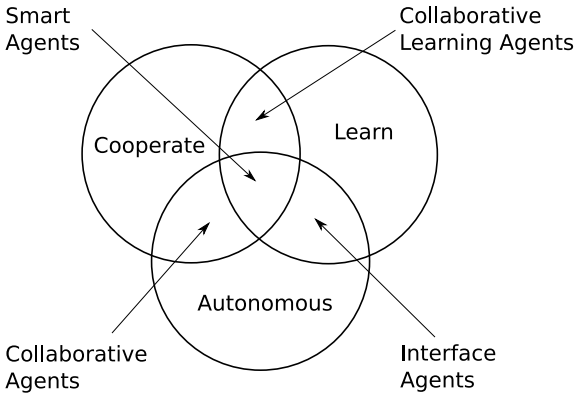


**Figure 1.1.:** A classification of agent types according to the three attributes: *cooperativeness*, *learning abilities* and *autonomy*. [Nwana, 1995]

### 1.1.1. Classification of Agent Types

Agents can be classified according to a variety of properties. Therefore, no generally accepted reference classification exists. We refer to a typology of agents established at the British Telecom laboratories [Nwana, 1995].

Three basic agent attributes serve as starting point for the classification. Agents are *autonomous* and they may *learn* and *cooperate*. Combinations of these properties lead to four agent types illustrated in figure 1.1. Following this systematisation, we do not deal with *collaborative learning agents* because they violate the autonomy condition of agenthood.

Up to now, this classification only deals with *how agents are*. If we enrich it with *what agents do*, this results in the following seven agent types. [Nwana, 1995]

**Collaborative agents** have social abilities. They are responsive and pro-active and can operate in open and time-constrained multi-agent environments.

**Interface agents** follow the metaphor of a *personal assistant*. They emphasise learning for optimal support of user interaction and collaboration.

**Mobile agents** can move around inside their environment.

**Information/Internet agents** connect to various distributed information sources and aggregate collected data.

**Reactive agents** do not necessarily require any internal (symbolic) representation of the world. They act in a stimulus-response manner. Intelligent behaviour emerges from interaction with the environment and other agents.

**Hybrid agents** combine features of different agent types, aiming to get the best out of all approaches mentioned above.

**Smart agents** would follow the stronger notion of agency.

Based on these definitions, our system core deals with hybrid agents. More specifically, it contains a mix of collaborative and reactive agents.

### 1.1.2. Agent Environments

The term *environment* is used in a twofold but related sense. In the following classification, the notion is used in terms of the world in which an agent resides. Later on, it additionally gets the meaning of available tools and mechanisms (infrastructure) for acting in an environment in the former sense.

Stuart Russell and Peter Norvig propose the following classification of environments [Russell and Norvig, 2003, p.41].

**Fully observable vs. partially observable:** In a fully observable environment, the agent's sensors detect all relevant information to decide about the next action. In contrast, in a partially observable environment agents must maintain an internal state to keep track of the changing world.

**Deterministic vs. stochastic:** A deterministic environment's next state is fully determined by the current state as perceived by the agent and the agent's action. In a stochastic environment this is not the case, again from the agent's point of view.

**Episodic vs. sequential:** In episodic environments, future actions do not depend on action taken in earlier episodes, which thus compartmentalise the overall evolution of the world. In a sequential environment, any decisions taken previously may influence all succeeding decisions.

**Static vs. dynamic:** Dynamic environments change even if the agent does not do anything. Otherwise, an environment is called static.

**Discrete vs. continuous:** This distinction can be applied to the *state* of the environment, how *time* is handled and to the agent's *actions* and *percepts*. Sometimes discrete data is treated like continuous data; e.g. a video stream from digital camera.

**Single agent vs. multi-agent:** A multi-agent environment accommodates multiple agents. A single agent environment contains only one agent. This distinction often depends on the environmental modelling. Sometimes it is possible to model other entities as stochastically behaving objects and sometimes it is better to model them as agents.

The hardest case is a partially observable, stochastic, sequential, dynamic, continuous and multi-agent environment which is often referred to as *open*.

According to [Huhns and Stephens, 1999] multi-agent environments

- provide an infrastructure specifying communication and interaction protocols,
- are typically open and have no centralised designer and
- contain self-interested or cooperative agents that are autonomous and distributed.

Additionally, the environment should provide support for agent and service discovery, certain communication protocols, concepts like ontologies and security services and guidelines for how an agent looks like in this environment.

### 1.1.3. Multi-Agent Systems

Modelling an application as multi-agent system is adequate in situations where information is distributed [Jennings, 2000]. One reason for information distribution may be that information is inherently distributed. Another reason is that information may be spread across (legacy) sub-systems. Further, the system as a whole need not have one single global top-level goal. Instead, accomodating several, maybe even conflicting goals, may be necessary for a system to be considered as successful. Therefore, some notion of rationality is inevitable to deal with complex interrelations of information and sub-systems. Multi-agent systems are appropriate for design and implementation of distributed computing systems.

Several multi-agent frameworks exist to ease development of concrete multi-agent system applications. For an overview over several such frameworks please refer to [Bordini R.H. et al., 2006]. Each framework has its focus on a set of features which makes it the framework of choice for a certain class of applications. Despite the differences, systems share some general key concepts:

A multi-agent system

> *" . . . contains a number of agents, which interact with one another through communication. The agents are able to act in an environment; different agents have different 'spheres of influence', in the sense that they will have control over - or at least be able to influence - different parts of the environment. The fact that these spheres of influence may coincide may give rise to dependency relationships between the agents."*
>
> [Wooldridge, 2002, p.105]

### 1.1.4. Agent Communication

Agents cannot force other agents to do something, but they can perform actions which *influence* other agents. Influencing others can be done implicitly by modifying the environment or directly by communication actions. The notion of communication as action is based on *speech act theory*. This theory goes back to the work of John Austin [Austin, 1962] and extensions by John Searle [Searle, 1970]. In human communication, several utterances share the characteristics of actions because they change the state of the world as shared among the peers. These types of utterances are called *speech acts*. *Performative verbs* correspond to certain types of speech acts. *Request* and *perform* are examples for performative verbs. Performative verbs are used in agent communication to define several types of messages. Such message types have well defined semantics and there is no doubt about the meaning of performative verbs.

*Agent Communication Languages* (ACLs) are typically message based and split into message envelope and message content. The message envelope contains used content language, used ontology, the performative, sender and receiver. The message content is expressed in an own language, often resembling first-order logic.

This splitting is done to separate the semantics of the ACL from the semantics of message content [Huhns and Stephens, 1999]. Examples for such communication languages are the *Knowledge Query and Manipulation Language* (KQML) [Finin et al., 1997] and the closely related *FIPA Agent Communication Language* (FIPA ACL) [Foundation for Intelligent Physical Agents (FIPA), 2002a]. KQML uses the *Knowledge Interchange Format* (KIF) by default for encoding message content but is not restricted to it. FIPA ACL may use SL, SL-0 (a subset of SL), CCL, KIF or RDF content language (see [Pitt and Mamdani, 1999] for an early critique).

An *ontology* defines objects, concepts and their relationships. The 'vocabulary' of an ontology is used in message content languages. Agents that understand or use a certain ontology agree on the meaning of the vocabulary in that ontology.

Another important aspect of multi-agent systems closely related to communication is coordination which is discussed in greater depth in section 1.1.6.

### 1.1.5. The Social Context in Multi-Agent Systems

In multi-agent systems, agents get in touch either with effects of other agent's actions or directly with agent peers. Consequently, it is necessary to discuss multi-agent systems from a social point of view.

Groups of directly or indirectly interrelated agents form agent organisations or societies [Huhns and Stephens, 1999]. Members in societies play on or more out of specific *roles* defined in it. To each role, certain commitments are associated. An agent assuming a role acquires the role's commitments, to which it is bound. The decision to join a group is affects an agent's autonomy. Once a group is joined, the agent it is obligated to the rules of this group. Groups define the *social context* in which agents interact.

### 1.1.6. Agent Coordination

Richard Jennings proposes that coordination is

*" . . . the process by which an agent reasons about its local actions and the (anticipated) actions of others to try and ensure the community acts in a coherent manner."*

[Jennings, 1996, p.187]

In [Nwana et al., 1996, Jennings, 1996] we find several reasons for the necessity of coordination, including the ones outlined here .

**Chaos prevention:** Agents only have a local view and local knowledge of the system. They are not omniscient. Without coordination, conflicting decisions could be taken. This may lead to chaos. In a chaotic system goal-oriented problem-solving is not possible.

**Global constraints:** In most systems global constraints must be obeyed for a solution to be considered as valid, e.g. cost constraints. Coordination of agent behaviours is necessary to meet such global constraints.

**Dependencies between actions:** Dependencies between actions impose an ordering on action execution. This ordering can only be achieved via coordination.

**Distributed knowledge:** Knowledge is distributed across the system. To use all knowledge available in the multi-agent system, coordination is necessary.

**Efficiency:** Information gathered by one agent may be used by another agent and therefore redundant work can be avoided. Such redundancy avoidance due to coordination increases efficiency.

Further examples of coordination mechanisms are market mechanisms [Kaihara and Fujii, 2005] and coordination media, such as tuple spaces [Denti and Omicini, 1999] (Linda [Carriero and Gelernter, 1989] being a prominent example).

[Nwana et al., 1996] proposes a basic classification of coordination strategies. The following four categories are taken from this paper.

**Organisational structuring:** The system designer takes care of coordination via predefined roles, responsibilities and dependencies between agents. These relationships are often expressed as hierarchical structures. An exponent for this kind of coordination is the typical master/slave relationship.

**Contracting:** The concrete organisational structure is determined during run-time via contracting mechanisms, like in the popular contract net protocol [Smith, 1988]. In contrast to *organisational structuring* this approach is completely distributed.

**Multi-agent planning:** Multi-agent planning can be further differentiated into *centralised* and *distributed* multi-agent planning. In the centralised approach, a dedicated coordination agent collects all plans and constructs a global multi-agent plan. This plan is then analysed. Conflicts and dependencies are resolved. In the distributed approach, each agent maintains its own plans and models of other agents' plans. Through communication, agents exchange these plans and refine their individual plans and remove inconsistencies. Through this exchange mechanism, a global complete plan results out of individual plans.

**Negotiation:** Negotiation is a major coordination mechanism and is nearly always used in the three coordination techniques mentioned above. Negotiation is considered a category of its own, because it is studied also in different academic disciplines.

Andrea Omicini and Sascha Ossowski study common principles of different coordination models and extract their similarities from an abstract point of view in [Omicini and Ossowski, 2003]. The following remarks are based on their work.

Coordination can be considered as the management of dependencies between goals, actions and plans of agents, that are considered as entities, which need to be coordinated. Coordination is a two step process. First, it is necessary to detect dependencies and second, possibly several coordination actions have to be taken to satisfy the detected dependencies. The TÆMS framework explicitly models coordination requirements in the form of hierarchical task decompositions.[7] Resolving dependencies and actions for goal achievement often results in complex interwoven interactions of agents inside the multi-agent system. The set of all possible interactions of one agent is often called *interaction space*. Another approach to coordination is to consider an agent as a situated entity in an environment. For achieving a goal it is necessary to execute some actions. Actions can be considered as altering the environment. Consequently, it is possible to influence agents' interactions by engineering the environment. The provided agent infrastructure can be designed to diminish the size of interaction space. This is where the notion of *coordination as a service* [Viroli and Omicini, 2006] comes into

[7] For details on TÆMS framework please refer to section 1.2.

play. Multi-agent system infrastructures may provide different coordination services. Once the agent has committed itself to a certain coordination service it is bound to the service's rules and laws. This commitment to a coordination service enables specific kinds of run-time control over agent behaviour. Centralised coordination mechanisms often go hand in hand with design-time coordination. They are most appropriate for closed environments whereas decentralised mechanisms and run-time coordination are better suited for open environments.

In our work, we deal with multi-agent planning and contracting, and use the TÆMS framework to address *qualitative* (dependencies between tasks) as well as *quantitative* (quality description through probability distribution) aspects of coordination. Our approach is a multi-centric one where certain agents provide planning and scheduling services via TÆMS structures. FIPA interaction protocols[8] are used for negotiation and run-time definition of coordination structures. We do not want to change how FIPA compliant multi-agent systems are engineered. We rather want to provide an important additional coordination mechanism to complement the range of existing ones.

### 1.1.7. Subjective versus Objective Coordination

There are basically two ways to examine coordination in a multi-agent system [Omicini and Ossowski, 2003]. The first one is from an agent's point of view and called *subjective coordination*. An agent observes the behaviours of other agents as well as the evolution of the environment over time. This information is filtered and interpreted by the agent. The result corresponds to an approximation of the agents' interaction space. An agent's coordination actions within a multi-agent system are driven by its own perception and understanding of the other agents' behaviours, capabilities and goals, as well as the environment state and its dynamics. Richard Jennings' definition mentioned above (see section 1.1.6) closely resembles this viewpoint of coordination. The second class, called *objective coordination*, adopts a bird's eye view on multi-agent systems: An external observer tries to affect agent interaction in a way that the progression of several agents' actions gears towards accomplishing one or more of the observer's goals. The observer must have a priori knowledge of agents' aims, capabilities and behaviours, as well as a means to anticipate foreseeing of the global behaviour of the

[8] http://www.fipa.org/repository/ips.php3, last visited May 2, 2007.

multi-agent system to reach effective coordination over time from the observer's point of view. Affecting agents by objective coordination does not necessarily imply that agents are manipulated directly. For example, affecting an agent could be achieved by systematically modifying (limiting, removing,...) resources this or other agents rely on.

To summarise, subjective coordination deals with coordination from an agent's point of view *inside* the multi-agent system whereas objective coordination is about influencing interaction among agents and the environment from the *outside*. Subjective coordination focuses on individuals in a multi-agent system. Objective coordination has its emphasis on the multi-agent system as a whole.

Andrea Omicini and Sascha Ossowski state that

> *"... objective coordination would conceivably take on the form of a collection of suitably expressive coordination abstractions, provided as run-time coordination services by the agent infrastructure.*
>
> [...]
>
> *... coordination as a service, that is, coordination provided as a service to agents by the infrastructure through a run-time coordination abstraction."*
>
> [Omicini and Ossowski, 2003, p.191, p.192]

Objective and subjective coordination are not mutually exclusive; rather they are complementary. *Coordination as a service* combines objective with subjective coordination. Here, objective coordination is the explicit representation of coordination laws outside of agents while subjective coordination addresses an agent's internal usage of these coordination laws provided at run-time by the multi-agent system infrastructure.

## 1.2. A Framework for Task Analysis, Environment Modeling and Simulation (TÆMS)

Task Analysis, Environment Modeling and Simulation (TÆMS) is a formal, domain-independent modelling language to represent hierarchical task structures. TÆMS can be understood as a model of an agent's partial view of a distributed goal tree

[Lesser et al., 2004]. This modelling framework offers language constructs for describing multiple ways to achieve a certain goal. The two main abstractions of TÆMS are *task*s and *method*s. The root of a goal tree is called *taskgroup*. In TÆMS, structures of multiple taskgroups may exist. A taskgroup is further decomposed into sub-goals which can be again tasks or methods. Leaf nodes of the goal tree represent atomic actions and are always methods. Additionally, TÆMS provides support for *interrelationships* (IRs) between tasks that do not stand in a task decomposition relationship. Consequently, TÆMS structures are not tree structures. Rather, they are digraphs and represent heterarchical decompositions. An agent's impact on its environment is modeled via *resources*.

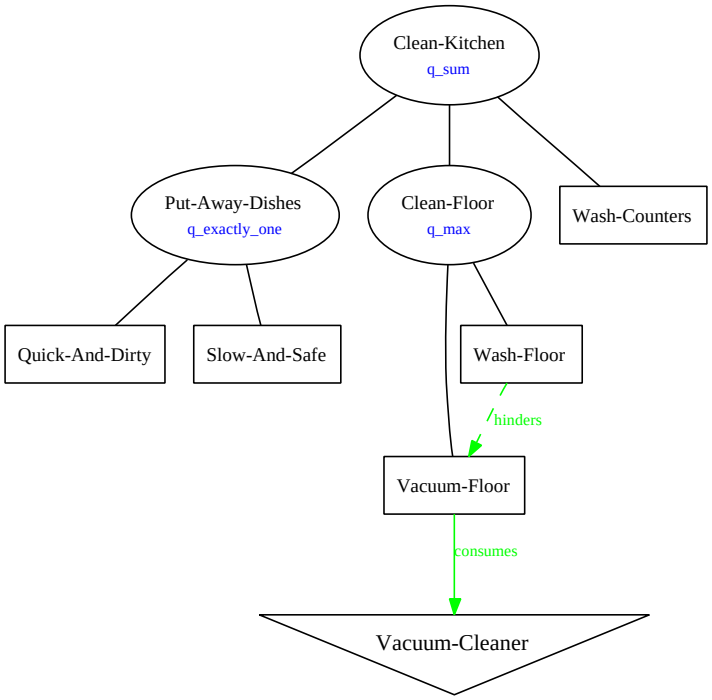


**Figure 1.2.:** A TÆMS example based on the small cleaning kitchen example from [Horling et al., 1999, p.71]

The most noteworthy property of TÆMS is not task decomposition. It is the fact that TÆMS addresses aspects of relevance for (soft) real-time requirements, handles uncertainty via probability distributions [Horling et al., 1999], and models worth-oriented domains, where goals can be reached to a different degree of success [Lesser, 1998, Zlotkin and Rosenschein, 1996]. Each method comes with a triple of discrete probability distributions describing quality, duration and cost of method execution. Therefore, TÆMS does not only support finding a way for achieving the goal. It rather

offers ways and means for finding certain paths to achieve a goal to a certain degree, depending e.g. on how much time is available for goal-achievement. The way of how to aggregate qualities from subtasks is defined by so-called *quality accumulation functions* (QAFs).

The main elements of TÆMS are [Jung, 2003]:

**Agent:** Definition of an agent. Multiple agent definitions in one TÆMS structure are possible.

**Taskgroup:** In principle, this is the root node of the goal tree; an agent's top-level goal.

**Task:** A task is an abstract activity that can be split into further tasks or methods.

**Method:** A method is an atomic, executable action. Method execution takes a certain amount of time and the result has certain cost and quality.

**Quality Accumulation Function (QAF):** QAFs define how quality of sub-tasks is aggregated to quality of the parent task. Additionally, they may impose ordering constraints on sub-tasks.

**Interrelationship (IR):** IRs are used to express influence of execution of one method on the execution of another method. IRs also connect methods with resources.

**Resource:** Explicit environmental modifications can be modeled as resources. Resources may be consumable or self-regenerating. Non self-regenerating resources can also be *produced*.

**Commitments:** Commitments are an agent's confirmations to do a certain action [Jennings, 1996]. *Local* commitments restrict an agent to certain tasks for the benefit of another agent, whereas *non-local* commitments are promises of other agents that help a certain agent.

TÆMS has two textual encodings called *TTAEMS* and *PTAEMS*. In section 2.1 we introduce a new XML-based encoding.

The quality accumulation functions defined in [Horling et al., 1999] are:

**q_min:** Task quality after accumulation is the minimum quality reached by the set of sub-tasks. All sub-tasks have to be successfully executed to reach a quality greater than 0.

**q_max:** Task quality after accumulation is the maximum quality reached by the set of sub-tasks. At least one sub-task has to be executed.

**q_sum** Task quality after accumulation is the sum of qualities reached by the set of sub-tasks. Not successfully executed tasks have quality 0.

**q_sum_all:** Task quality after accumulation is the sum of qualities reached by the set of sub-tasks. All sub-tasks have to be executed, no matter whether any of them fails.

**q_seq_min:** Task quality after accumulation is equal to q_min. Additionally, sub-tasks have to be executed in a specific order.

**q_seq_max:** Task quality after accumulation is equal to q_max. Additionally, sub-tasks have to be executed in a specific order.

**q_seq_sum:** Task quality after accumulation is equal to q_sum. Additionally, sub-tasks have to be executed in a specific order.

**q_seq_last:** Task quality after accumulation is equal to the last executed sub-task quality. Additionally, sub-tasks have to be executed in a specific order.

**q_exactly_one:** Exactly one sub-task must be executed. Quality after accumulation equals the quality of this task. If more than on task is executed, accumulated quality equals 0.

**q_last:** Task quality after accumulation is equal to the last executed sub-task quality. The order of execution does not matter.

**q_sigmoid:** This QAF is mentioned but not specified in any detail.

The development of TÆMS is closely related to the evolution of the *Java Agent Framework*[9] (JAF) which uses it as its main data structure for problem representation and communication, as well as to GPGP and DTC-scheduling, both being discussed in the next sections. For further details on TÆMS please refer to [Horling et al., 1999] and [Lesser et al., 2004]. For a critical discussion of TÆMS see [Jung, 2003, Jung and Petta, 2003, Jung and Petta, 2004].

[9] http://dis.cs.umass.edu/research/jaf/, last visited May 2, 2007.

## 1.3. Generalized Partial Global Planning (GPGP)

Generalized Partial Global Planning has its roots in *Partial Global Planning* (PGP) [Durfee and Lesser, 1991] used in *Distributed Vehicle Monitoring Testbed* (DVMT) [Lesser and Corkill, 1983]. The main idea is to enrich an agent's local knowledge for solving problems with knowledge of other agents. In PGP, an agent uses *partial* information about other agent's goals to construct a *partial global* view of the problem to be solved. This partial global view can be used by an agent for planning and scheduling actions to accomplish its own goals. The representation of information to share unfortunately is domain-dependent. The idea behind GPGP remains the same like in PGP, but GPGP:

> *"...tries to extend the PGP approach by communicating more abstract and hierarchically organized information, detecting in a general way the coordination relationships that are needed by the partial global planning mechanisms, and separating the process of coordination from local scheduling."*
>
> [Decker and Lesser, 1992, p.321]

TÆMS is the formal domain-independent framework for representation of coordination issues. GPGP assumes a local scheduler at each agent. Information for the scheduler can be represented by local and non-local commitments to tasks in the task structure or by altering and extending the local task structure. The main field of application for GPGP and TÆMS is worth oriented [Lesser et al., 2004].

GPGP defines the following five coordination mechanisms [Decker and Lesser, 1995] to provide more information to the local scheduler for construction of better schedules.

#### Updating Non-Local Viewpoints

The detection of coordination relationships is highly domain specific and cannot be handled in a generic fashion. But the result of detection always is a TÆMS structure describing the local beliefs of an agent. Updating non-local viewpoints means finding other agents with overlapping beliefs and further gathering information about them. For the present purposes, we can equate a belief with a task in the local TÆMS structure. An agent $\alpha$ wants to update its tasks and it finds an agent $\beta$ with a overlapping task $t$. Let us assume that agent $\beta$ has further tasks associated with task $t$. According to its willingness,

agent $\beta$ can tell agent $\alpha$ about all, some or none of the associated task with task $t$. Agent $\alpha$ may include these additional tasks in its task structure and thus has updated its local subjective viewpoint with non-local information.

### Communicating Results

Three policies for an agent to communicate results exist in GPGP:

- Communicate only results to satisfy commitments with other agents;
- Like the first policy, but communicate also the final result associated with a task group;
- Communicate all results obtained.

### Handling Simple Redundancy

Redundancy may occur if multiple agents provide the same method. If multiple agents want to execute a method, the simplest way to deal with this problem is to randomly choose an agent and let it execute the method. After method execution has finished, the result must be sent to all other agents also wanting to execute this method. This approach does not provide any support for load balancing. It also does not take any quality considerations into account. General redundancy detection is domain-dependent and resolution is difficult.

### Handling Hard Coordination Relationships

Hard coordination relationships apply a strict temporal ordering to the execution of methods or tasks. Since this mechanism is a pro-active one and only implemented for the *enables* relationship, the challenge is to find the earliest possible start time for a task $t_1$ under the condition that the dependent task $t_2$ still has positive quality. The calculated value for the start time of $t_1$ is used for a local and non-local commitment to $t_1$. Committing to the earliest start time of $t_1$ allows the agent executing $t_2$ more slack time. The local commitment has low negotiability.

### Handling Soft Coordination Relationships

In principle, soft coordination relationships are handled in the same way as hard coordination relationships. The only difference is that local commitments have high negotiability.

## 1.4. VIE-CDS

VIE-CDS [Jung, 2003, Jung and Petta, 2004] implements the *Design-to-Criteria*[10] (DTC) scheduling algorithm developed at the Multi-Agent Systems Laboratory[11] of the Department of Computer Science of the University of Massachusetts Amherst.

DTC scheduling offers a flexible and domain-independent approach to task scheduling. DTC scheduling constructs an ordering of methods based on TÆMS structures that suffices

- Restrictions of task decomposition,
- Temporal constraints,
- Commitments of agents,
- Resource usage, and
- Relative preferences for solutions according to quality, cost and duration.

These relative preferences distinguish DTC scheduling from other scheduling strategies. DTC scheduling does not only find valid schedules. It is possible to tell the algorithm to prefer solutions with a certain trade-off between quality, cost and duration to reach a goal. In [Wagner et al., 1998] the *slider metaphor* is introduced to define the relative importance of quality, cost and duration via so called *importance sliders*. Another feature of DTC scheduling is that no backtracking is involved in the scheduling process. Figure 1.3 illustrates control flow in the DTC scheduling algorithm.

In [Jung, 2003] Bernhard Jung discusses several shortcomings and inaccuracies of TÆMS design and the associated JAF-DTC scheduler. He proposes a number of clarifications and related approaches, which are implemented in the VIE-CDS[12] scheduler.

VIE-CDS is written in the Java programming language. It uses several features of object oriented programming and dynamic binding to realise a modular and extensible software architecture. It offers full support for PTAEMS and TTAEMS encodings of TÆMS structures. Since DTC uses TÆMS as representation language for coordination

[10] http://dis.cs.umass.edu/research/dtc/, last visited May 2, 2007.
[11] http://dis.cs.umass.edu/, last visited May 2, 2007.
[12] http://www.ofai.at/research/agents/projects/viecds/, last visited May 2, 2007.

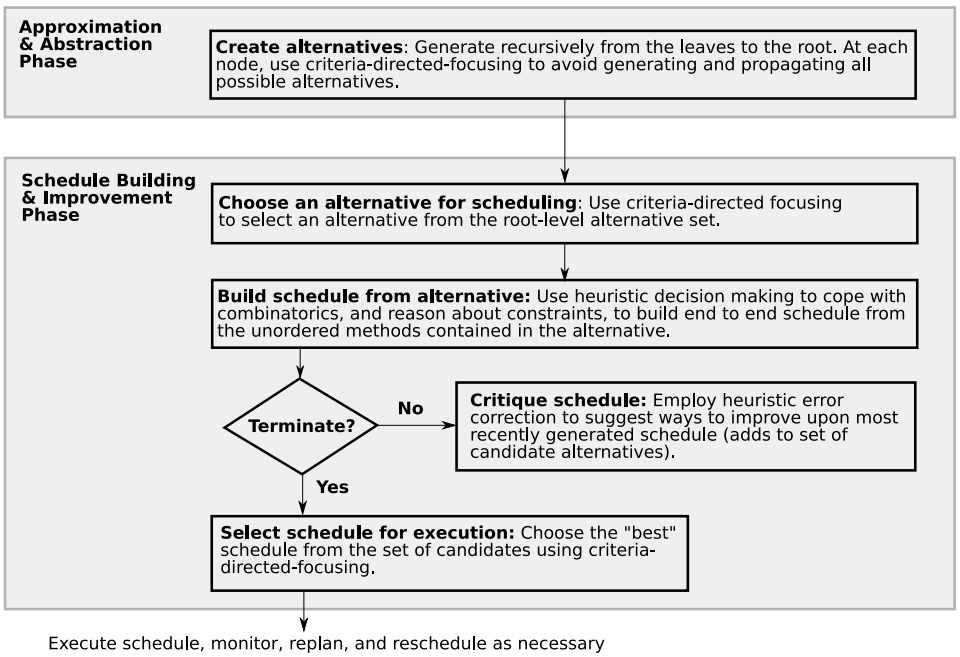


**Figure 1.3.:** High-level control-flow in the DTC scheduling algorithm, [Wagner et al., 1998, p.95]

issues, VIE-CDS also provides a TÆMS implementation. VIE-CDS offers a command-line and a graphical user interface for interaction. In chapter 2 we address internals of VIE-CDS.

## 1.5. Java Agent Development Framework (JADE)

JADE is a FIPA 2000 compliant [Bellifemine et al., 2001] software framework for development of agent-based applications. It can be obtained from the JADE homepage[13]. JADE's main purpose is to ease development of multi-agent systems by

- Following FIPA standards,
- Being designed as a distributed agent platform,
- Providing ready to use implementations of agent interaction protocols,
- Providing graphical user interfaces for platform management,
- Providing a framework for agent construction,

[13]http://jade.tilab.com, last visited May 2, 2007.

- Hiding complex intra- and inter-platform communication issues behind a simple API and
- Using Java as the programming language of choice.

JADE was introduced at the time FIPA 97 was the current reference architecture. At the time of writing, JADE has reached version 3.4.1 and is a mature, FIPA 2000 compliant agent development framework. JADE is free software and licensed under the terms and conditions of LGPL version 2. Telecom Italia Lab (TILAB)[14] is the copyright holder. Since May 2003, the *JADE Board* consisting of TILAB, Motorola[15] and Whitestein Technologies AG[16] supervises management of the JADE project.

In this master's thesis we refer to JADE version 3.4.1 released on November 17, 2006.

### 1.5.1. FIPA Specifications

The *Foundation for Intelligent Physical Agents* (FIPA)[17] is an *IEEE Computer Society Standards* organisation. It consists of several companies and organisations aiming to promote agent-based technologies. FIPA does not define technologies at implementation level. Instead, it specifies general, domain-independent requirements for agent-based systems.

At their core FIPA 97 specifications identify three necessary agent roles. The *Agent Management System* (AMS) is responsible for control over access to the agent platform and its use. The AMS is responsible for maintaining a directory of all agents inside the agent platform. The *Agent Communication Channel* (ACC) provides the default message routing service. The ACC must speak the *Internet Inter ORB Protocol* (IIOP) to enable communication with other FIPA compliant platforms. The *Directory Facilitator* (DF) is an agent that provides a yellow pages service. An agent is defined as a fundamental actor that aggregates several service capabilities to form a unified and integrated execution model. [Bellifemine and Rimassa, 2001]. These fundamentals are still valid for the FIPA 2000 specification. Since FIPA 2000 is more elaborate and extensive than FIPA 97, the definition of ACC has moved from agent management specification to a

[14]http://www.telecomitalialab.com/, last visited May 2, 2007.
[15]http://www.motorola.com/, last visited May 2, 2007.
[16]http://www.whitestein.com/, last visited May 2, 2007.
[17]http://www.fipa.org, last visited May 2, 2007.

separate message transport specification. Figure 1.4 illustrates the FIPA 2000 agent platform reference model.

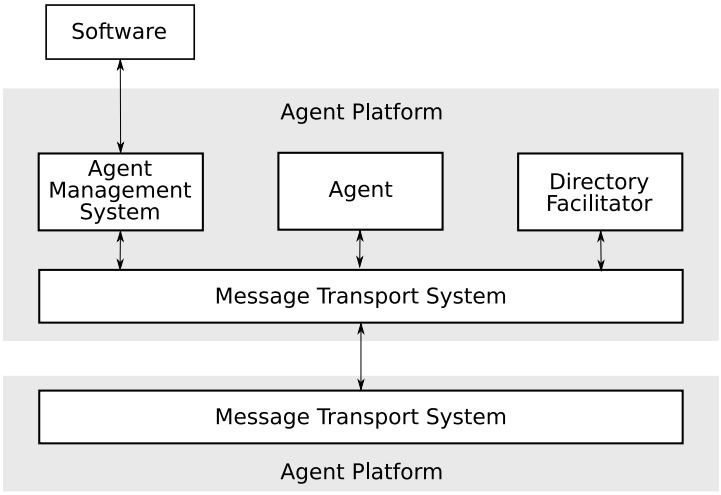


**Figure 1.4.:** FIPA agent platform reference architecture [Foundation for Intelligent Physical Agents, 2004, p.5].

Additionally, FIPA standardises general conditions for the communication infrastructure. The *Agent Communication Language* (ACL) resembles KQML but has formally defined semantics. For inter-platform communication FIPA defines a textual encoding of messages; no specific transportation mechanisms are specified for intra-platform communication. FIPA defines the usage of certain *Content Languages* (CL) like *Knowledge Interchange Format* (KIF), *Resource Description Framework* (RDF) and *Semantic Language* (SL), but does not constrain the programmer to them. Common forms of inter-agent conversations are supported by so called *Interaction Protocols* which are patterns of messages exchanged by two or more agents.

### 1.5.2. Platform Architecture

The JADE distributed agent platform is built around the concept of containers. Each container is run by one Java Virtual Machine (VM). A container is a fully fledged runtime environment for agents. The agent platform consists of several containers where different containers may reside on different hosts. Inside a container several agents can exist. The *main-container*, also called front-end container, is a special container. It maintains the *Agent Global Descriptor Table*, which contains references to all agents on

the platform, and the *Agent Container Table*, which stores a reference to each container in the platform. When the main-container boots, it automatically instantiates ACC, AMS and DF services. Inside a container communication occurs via Java event mechanisms. Basically, this is done by exchanging references to Java objects. Inter-container messaging uses Java *Remote Method Invocation* (RMI). In principle, a container is a RMI server and the main-container is the RMI registry. Figure 1.5 shows a schema of a JADE agent platform distributed over several hosts.

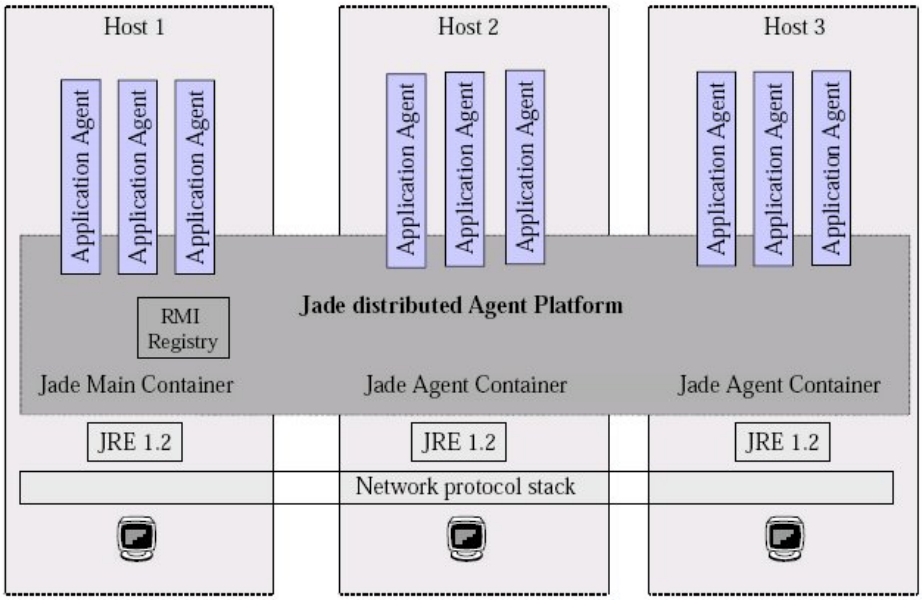


**Figure 1.5.:** Schema of a JADE platform distributed over several containers on different hosts. [Bellifemine et al., 2006a, p.8]

At the expense of communication overhead it is possible to use main container replication. This mechanism allows multiple main-containers to exist, but one is always the master main-container (and the others are slave main-container). Agent registration is possible at each main-container. If one main-container should crash another main-container takes over its functionality (even in the case of a master main-container fault). Figure 1.6 contrasts a single main-container layout with a main-container replication layout. JADE makes extensive use of caching techniques to avoid iterated look-ups for agent or container references [Bellifemine and Rimassa, 2001, Bellifemine et al., 2006b].

Generally speaking, JADE tries to map all FIPA concepts to the object oriented paradigm. Messages, message content, agents, behaviours, ontologies, transportation mechanisms, etc., are represented by native Java objects. If JADE needs to communicate with agents

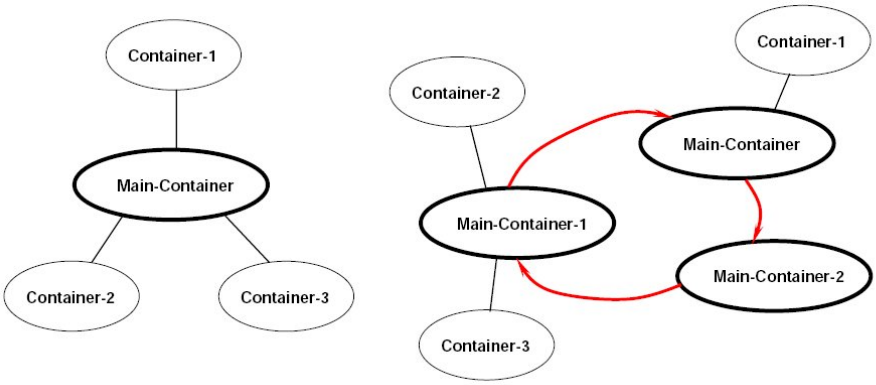


**Figure 1.6.:** The connected graph to the left illustrates a single main-container layout, whereas the right one shows a fault tolerant replicated layout. [Bellifemine et al., 2006b, p.20]

or platforms that cannot deal with Java objects, it automatically translates Java objects to a representation the other side understands, presumed that JADE knows about an appropriate encoding. Services like the automatic translation and other platform specific services can be extended by plug-in mechanisms called *kernel-level services* which must not be confused with the services provided by agents.

### Agent Model

In principle, JADE agents are Java threads inside a VM representing a container. Agent autonomy and social abilities push for concurrency inside an agent. Due to the fact that only a limited number of threads per VM are possible, JADE introduces the *Behaviour* abstraction. A behaviour represents an ability of an agent which needs to be executed in parallel to other behaviours. Behaviours are the major building blocks of agent functionality. Even for sending and receiving messages behaviours are used. Incoming messages are stored in the agent's message queue. When a receiver behaviour is scheduled it can read a message from the message queue. Behaviours are scheduled cooperatively. This means that there is no support for pre-emption: behaviours are self-dependent for returning control to the scheduler. To preserve the illusion of concurrency, it is inevitable that any slice of behaviour execution takes only a short amount of time. At return of control to the scheduler, a behaviour can indicate when it should not have finished its work. This behaviour then becomes re-scheduled. In that case, the behaviour itself is responsible for saving and restoring its internal state. JADE offers the possibility to split behaviours

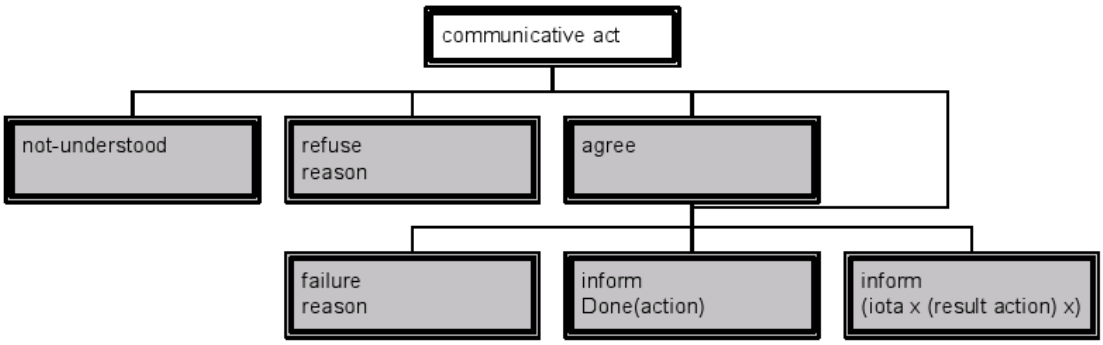


**Figure 1.7.:** JADE's implementation abstraction for FIPA interaction protocols, [Bellifemine et al., 2006a, p.35]. Each interaction protocol following the *Achieve Rational Effect* pattern may reply to a message with a certain performative with a not-understood, refuse or agree message. After the agree message a failure or inform message must follow. JADE allows by-passing the agree message and sending a failure or inform message in direct response.

into sub-behaviours. In addition, it is still possible to use Java threads inside agents, but this limits the maximum number of concurrent agents, and it is the programmer who must deal with synchronisation of agent and threaded behaviours. For further details on agent design please refer to [Bellifemine et al., 2006a].

### Interaction Protocols

Since all FIPA interaction protocols have a resembling structure and are geared to achieve a rational effect, JADE provides a common implementation abstraction. The initiator role is represented by behaviour *AchieveREInitiator*, the behaviour *AchieveREResponder* serves the responder role. Figure 1.7 illustrates the procotol structure used by JADE. The initiator sends a message containing a certain performative. The responder may answer with `NOT-UNDERSTOOD`, `REFUSE` or `AGREE`. After having agreed, sending an `INFORM` or a `FAILURE` message is mandatory. Against FIPA specifications, the `AGREE` state is optional. Instead, the agent may send instantaneously an `INFORM` or a `FAILURE` message.

# 2. Evolution of VIE-CDS

The main goal of this master's thesis is the integration of criteria-driven scheduling with FIPA-compliant multi-agent infrastructures. During the analysis of VIE-CDS concerning our DTC planning and scheduling integration approach, it turned out that having a new TÆMS encoding, abstracting away the strong relationship of TTAEMS to JAF would be advantageous. This section documents the design of our new encoding called XTAEMS and its relation to the TTAEMS encoding, as well as the integration of XTAEMS with VIE-CDS.

VIE-CDS has been designed by Bernhard Jung with the design goals of flexibility and modularity [Jung, 2003, Jung and Petta, 2004]. Even so, for the present efforts some refactoring of source code was necessary to ease extensibility concerning various input and output formats. In section 2.1, the new XML-based XTAEMS encoding of TÆMS is introduced. Section 2.2 presents the related updates to the internal structure of VIE-CDS, including the parser API (see section 2.2.1) and the serialisation API (see section 2.2.2). We end this chapter with an evaluation of the new XTAEMS parser.

## 2.1. A New Encoding for TÆMS

In section 1.2 we have given an introduction to TAEMS. We now present the encoding for TAEMS structures published in [Horling et al., 1999] and also used in [Jung, 2003], followed by an explanation of the reasons that led us to develop a new XML-based encoding, XTAEMS.

The TTAEMS encoding features a LISP-like syntax [Horling et al., 1999]. As in LISP nearly everything is wrapped with a pair of parentheses. Most elements in TTAEMS follow roughly this attribute/value form:

```
(field_name field data)
```

The token `field_name` must be of data type *symbol*. It describes the content of the element, whereas `field data` is the actual data or value associated with that name. Generally, in TTAEMS the following basic data types are defined:

**Symbol:** A word composed of the characters [A-Za-z0-9] and the symbols "_" or "-". The word must not contain whitespaces.

**Agent Symbol:** This data type follows the same rules like symbol. Additionally, it should match the name of an agent in the multi-agent system.

**Predefined Symbol:** This data type follows the same rules like symbol, but the list of allowed symbols is predefined and must be specified explicitely by the field.

**Integer:** An integer value, e.g. -27

**Float:** A float value, e.g. 0.333

**Distribution:** A series of pairs of floats. The first value of a pair represents a data point and the second one the associated probability.

**List:** A series of whatever type is given seperated by whitespaces.

**Special:** The data type is field-specific and must be defined seperately.

**None:** A field of this type has no data.

A TTAEMS description consists of a number of objects, where each object is described within a self-contained `(spec_* ... )` block, and has a label unique within the structure [Horling et al., 1999]. As in LISP, a line starting with a semicolon is treated as a comment line. For further details on TTAEMS please refer to [Horling et al., 1999].

Since we wanted an encoding that could be managed with state of the art standard technologies and tools, we developed an XML-based encoding for TÆMS structures, called XTAEMS. We identify the following advantages for using XML over the LISP-like format:

- The XML format is standardised by the *World Wide Web Consortium*[1]. This facilitates tool development because of the documented well-known syntax.

[1] http://www.w3.org/, last visited May 2, 2007.

- XML comes with the well defined *Document Object Model* (DOM) which defines an in-memory model for XML data. This eases manipulation of TÆMS structures.
- The *Simple API for XML* [2] (SAX) is a de facto standard for high-performance parsing. It supports all features necessary for parsing TÆMS structures encoded in XTAEMS.
- XML can be easily transformed into other (text based) formats with so called XSLT-stylesheets.
- XML-namespaces offer the possibility to embed XTAEMS into other XML-based formats such as the *Resource Description Framework*[3] (RDF) or the *Web Ontology Language*[4] (OWL).
- Well tested and really high-performance tools for manipulating XML exist. We use Apache's *Xerces*[5] parser and *Xalan*[6] XSLT stylesheet processor.

XTAEMS is designed to be as close as possible to TTAEMS. The following rules have been used to derive the XML encoding of XTAEMS from the LISP-like TTAEMS encoding.

- XTAEMS has a document element called `taems`.
- The encapsulating structure of of TTAEMS was preserved.
- Each TTAEMS-token starting with `spec_#name#` is mapped to an XTAEMS tag called `#name#`, e.g. `spec_task` becomes `<task>`. `#name#` is a placeholder for special token types named equally in TTAEMS and XTAEMS.
- Each TTAEMS-token having only one argument is added to its parent token as an attribute. This applies to the TTAEMS data types *symbol*, *agent symbol*, *integer* or *float*. Listing 2.1 provides an example for such a conversion.
- TTAEMS `task_group`s are treated like `tasks` (see below).
- `supertasks` have no corresponding tag in XTAEMS (see below).
- The leading `q_` of quality accumulation function names is removed.

[2] http://www.saxproject.org/, last visited May 2, 2007.
[3] http://www.w3.org/RDF/, last visited May 2, 2007.
[4] http://www.w3.org/TR/2003/WD-owl-ref-20030331/, last visited May 2, 2007.
[5] http://xerces.apache.org/, last visited May 2, 2007.
[6] http://xalan.apache.org/, last visited May 2, 2007.

```
(spec_task
        (agent anyAgent)  becomes   <task agent="anyAgent"/>
)
```

**Listing 2.1:** Example conversion of a TTAEMS token with a single argument (left) into XTAEMS representation (right).

```
                                <subtask ref="t1"/>
(subtask t1 t2 t3)  becomes     <subtask ref="t2"/>
                                <subtask ref="t3"/>
```

**Listing 2.2:** Example conversion of a TTAEMS subtask token (left) to XTAEMS (right).

- The TTAEMS-token `label` is renamed to `id`, because DOM allows searching for XML tags according to their id tag.
- The TTAEMS-token `subtasks` is mapped to multiple XTAEMS tags named `subtask` with an attribute named `ref`. Listing 2.2 shows an example of this conversion.
- The data type *distribution* of TTAEMS is represented by a tag with the name of the distribution, e.g. `quality`. Each point/probability pair becomes a `distribution` tag with an attribute called `point` containing a data point and an attribute called `probability` containing the probability at this point. Listing 2.3 gives an example for this conversion.
- The Attribute `density` in `distribution` is replaced by `probability`, because TÆMS deals with discrete probability distributions.

```
                                          <quality>
                                             <distribution point="0"
                                               probability="0.7"/>
(quality 0 0.7 0.5 0.2 1 0.1)   becomes   <distribution point="0.5"
                                               probability="0.2"/>
                                             <distribution point="1"
                                              probability="0.1"/>
                                          </quality>
```

**Listing 2.3:** Example conversion of data type *distribution* from TTAEMS (left) to XTAEMS (right).

- Optional TTAEMS-tokens also remain optional in XTAEMS.
- Agent definitions become optional (see below).
- TTAEMS-tokens of data type *none* become an attribute of the parent tag with the name of this token. The attribute's value is `true` if this token is present in TTAEMS representation. Otherwise it is `false` or not even added as a XTAEMS attribute.
- For each TTAEMS-token including an underscore, the underscore is removed and the letter right after the underscore is changed to upper case.
- Outcomes do not have names. Their order is important. References to outcomes are made according to their rank. Counts starts with 0. Listing 2.4 exemplifies this conversion on a method with two outcomes and a `facilitates` interrelationship referencing one of this outcomes.
- If no *model* is specified in an interrelationship then `duration_independent` is assumed.
- `outcome` tags are not encapsulated by `outcomes` like indicated by the rules above. They are direct children of the `method` tag.

The most prominent changes are the removal of `task_group` and `supertasks` and setting the `agent`-field to be optional. `task_group` has been removed because the only difference to `task` is that no `subtask` relation to this `task` exists. Through `subtask` relations all `task_group`s can be calculated. The same idea has prompted the removal of `supertasks`. Through the `subtask` relationship all `supertasks` can be calculated. The `agent` field has been removed because we assume that owners of certain methods can be resolved via something similar to a yellow pages service provided by the underlying multi-agent infrastructure.

XTAEMS features human readability, but we argue this can be achieved better through graphical representations. In contrast to TTAEMS, XTAEMS is not intended to be used as internal data structure for multi-agent applications. We rather conceive XTAEMS as an abstract language to describe a problem's inherent structure.

In the TÆMS white paper ([Horling et al., 1999]) in section 5, three different ways of representing coordination and negotiation issues are discussed.

```
<method id="m1">
    <outcome probability="0.7">
        <quality>
            <distribution point="0" probability="1"/>
        </quality>
        <cost>
            <distribution point="0" probability="1"/>
        </cost>
        <duration>
            <distribution point="0" probability="1"/>
        </duration>
    </outcome>
    <outcome probability="0.3">
        <quality>
            <distribution point="10" probability="1"/>
        </quality>
        <cost>
            <distribution point="10" probability="1"/>
        </cost>
        <duration>
            <distribution point="10" probability="1"/>
        </duration>
    </outcome>
</method>

<facilitates id="fac1" from="m1" to="m4" fromOutcome="0">
    <quality>
        <distribution point="1" probability="1"/>
    </quality>
    <cost>
        <distribution point="1" probability="1"/>
    </cost>
    <duration>
        <distribution point="1" probability="1"/>
    </duration>
</facilitates>
```

**Listing 2.4:** Method `m1` has two possible outcomes. The facilitates interrelationship `fac1` references the first of these outcomes.

**heterogeneous agent fields** The `agent` field is used to determine who is responsible for this task or method.

**non-local methods** This is a further abstraction in TÆMS and a possible point of extension in the TÆMS structure. An agent has knowledge about the existence of this method but not about who is providing it.

**explicit representation** Coordination issues are directly modelled as methods being scheduled like other methods. This has the advantage of clarity and offers the possibility to express special characteristics of coordination.

Because XTAEMS has been developed with our approach of integrating VIE-CDS with JADE in mind we opt for a mixture of non-local methods and explicit representation. We assume that an agent is able to perform a method unless it marks the method as *non-local* in its own TÆMS structures. We also assume that a yellow pages service provides our coordination mechanism with information about who is responsible for which method. Also for local scheduling without *non-local* methods an *agent* field is not needed. It should be quite clear that only the scheduling agent's abilities are referred to. Resolving redundancy and interrelationships is done by our coordination module which alters the TÆMS structures provided by the agents. It uses domain knowledge to explicitly insert coordination relationships. Our coordination module is discussed in great depth in chapter 3.

The appendix to the TÆMS White Paper defines a TTAEMS encoding for schedules. Schedule representation has also been mapped to XTAEMS using the rules above. Schedules may reside in a separate file or be embedded into a TÆMS representation. A TTAEMS encoding for commitments offering a method to represent the quantitative aspects of negotiation is also proposed in the White Paper's appendix. We leave this uncovered, because we want to use dedicated multi-agent system communication infrastructure facilities for negotiation.

We propose `.xtaems` as standard file extension for XTAEMS-encoded TÆMS structures and `.sxtaems` for XTAEMS encoded schedules. We also provide a XSLT stylesheet for easy conversion from XTAEMS to TTAEMS. If there are methods or tasks without an agent reference, this stylesheet defines an agent called *anyAgent* and assigns all these methods and tasks to this agent. `Supertask` relationships, `task_group`s and `outcome` names are also calculated. `Outcome` names consist of `Outcome_` followed by the rank of the outcome. Appendix A contains a number of XTAEMS examples.

## 2.2. New Structure of VIE-CDS

VIE-CDS up to revision 0.2 is composed of a solid kernel for TÆMS in-memory structures, parsing and DTC-computations. Interaction is done via a command-line interface written in BeanShell[7] and shell scripts. Additionally, VIE-CDS has a graphical user interface. In our work, the BeanShell dependency has been removed. BeanShell scripts are replaced by additional classes in VIE-CDS and the command line parsing has been extended using JSAP[8]. VIE-CDS is intended to be delivered as a single jar-file to ease integration. In conjunction with the new parser and serialisation interface these changes induce the need for some code refactoring. Figure 2.1 illustrates the new package structure of VIE-CDS.

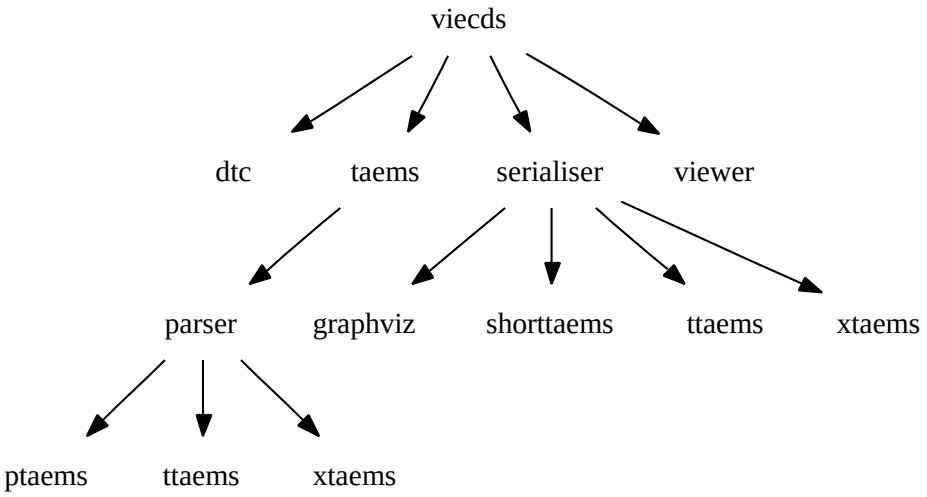


**Figure 2.1.:** The new VIE-CDS package structure

Our new packaging arranges classes into semantically coherent groups. The package `viecds` is the top-level package containing all sub-packages VIE-CDS consists of. `VieCDS` is the one and only class in this package. This class contains the `main`-method and provides the command line interface for VIE-CDS. Package `viecds.dtc` encapsulates all classes specific to the DTC-algorithm. Package `viecds.taems` includes all classes for in-memory representation of TÆMS structures. Package `viecds.serialiser` contains classes and sub-packages for serialising the in-memory TÆMS representation to various

[7] http://www.beanshell.org/, last visited May 2, 2007.
[8] http://www.martiansoftware.com/jsap/, last visited May 2, 2007.

textual formats. Package `viecds.viewer` provides the graphical user interface already known from the former VIE-CDS.

### 2.2.1. Parser API

The Parser API resides in package `viecds.taems.parser`. It consists of the following seven classes:

**Parser** is an interface defining the method `parse()` with return type `TAEMS`. Every class implementing this interface can be used as a parser. We strongly advise to implement parsers as sub-packages of this package.

**ParserFactory** is the only recommended way to create Parser instances. It provides the `createTAEMSParser()` method. This method expects a `ParserType` object as first parameter and a `File`, `InputStream` or `String` object containing the TÆMS structure as second parameter.

**ParserType** is an enumeration. It provides the three fields TTAEMS, PTAEMS and XTAEMS, representing all parsers available in the system. This information is also used in the usage message of the command-line interface.

**ParseException** is a class created by the *Java Compiler Compiler*[9] (JavaCC). As this and the following three classes are shared by all created parsers, we have placed it here to avoid duplication.

**SimpleCharStream** is an implementation of the interface CharStream, where the stream is assumed to contain only ASCII characters.

**Token** is also a class created by JavaCC. It describes the input token stream.

**TokenMgrError** represents a lexical error.

The TTAEMS and PTAEMS parsers are unchanged over the former version of VIE-CDS and have only been rebuilt with JavaCC version 4.0. They have been moved to the packages `viecds.taems.parser.ttaems` and `viecds.taems.parser.ptaems`. The PTAEMS parser basically runs a pre-processor on the input and expands it to normal TTAEMS. Then the standard TTAEMS parser is started on expanded input. The XTAEMS parser is discussed in detail in section 2.3.

[9] https://javacc.dev.java.net/, last visited May 2, 2007.

### 2.2.2. Serialisation API

The idea behind the new serialisation API is to define a standard interface for the transformation of in-memory TÆMS structures to various output formats. This API resides in the package `viecds.serialiser`. It is strongly recommended to implement concrete serialisers as sub-packages.

The following five classes form the serialisation API:

**ISerialisationResult** is an interface representing the result of a serialisation process. It defines a `write()` method for a `PrintStream` or `File` object. Its main purpose is to offer a generalised way of writing the serialisation result to an operating system resource.

**ISerialiser** is the interface all serialisers must implement. It defines a `serialise()` method for a `TAEMS`, a `SetOfSchedules` or a `Schedules` object with return type `ISerialisationResult`.

**SerialisationException** is the exception expected to be thrown if an error during the serialisation process occurs.

**SerialiserFactory** is the only recommended way to instantiate Serialisers. It defines the `createSerialiser()` method for an object of type `SerialiserType`. This method returns an serialiser object corresponding to the `SerialiserType`.

**SerialiserType** is an enumeration providing the fields XTAEMS, TTAEMS, SHORT and GRAPHVIZ. This enumeration describes all available serialisers and is used for the usage message of the command-line interface.

#### XTAEMS Serialiser

The XTAEMS serialiser is implemented in the package `viecds.serialiser.xtaems`. It consists of the classes `XTAEMSSerialiser` implementing `ISerialiser` and `XTAEMSResult` implementing `ISerialisationResult`. This serialiser iterates over each in-memory TÆMS object and builds a XTAEMS conforming XML representation using DOM and the serialisation API from Apache.

### TTAEMS Serialiser

TTAEMS Serialiser is implemented in package `viecds.serialiser.ttaems`. It consists of the class `TTAEMSSerialiser` implementing `ISerialiser` and `TTAEMSResult` implementing `ISerialisationResult`. It does not introduce any new functionality but uses the TTAEMS export methods of the former VIE-CDS.

### Short-TAEMS Serialiser

The Short-TAEMS serialiser is implemented in package `viecds.serialiser.shorttaems`. It consists of classes `ShortTAEMSSerialiser` implementing `ISerialiser` and `ShortTAEMSResult` implementing `ISerialisationResult`. Short-TAEMS is a human readable schedule-only representation introduced with VIE-CDS. Short-TAEMS outputs only the best rated schedule. It contains the number of methods, the methods themselves with their abstract probabilistic quality descriptions and a quality description of the whole schedule. For an example, see appendix B.1.

### Graphviz Serialiser

The Graphviz[10] serialiser is implemented in package `viecds.serialiser.graphviz`. It consists of the classes `GraphvizSerialiser` implementing `ISerialiser` and `GraphvizResult` implementing `ISerialisationResult`. Graphviz output is not intended for direct use. Graphviz is an open source graph description language for use with several graph layouting tools developed by AT&T. A command-line tool like `dot` takes this description as input and produces (hierarchical) drawings of directed graphs in various graphical formats. This schedule serialiser takes the three best rated schedules into account. For examples, see appendix B.2.

## 2.3. XTAEMS Parser

The XTAEMS parser is implemented in package `viecds.taems.parser.xtaems`. It consists of only one class called `XTAEMSParser` which extends `DefaultHandler` from `org.xml.sax.helpers` and implements our parser interface and `ErrorHandler` from

[10] http://www.graphviz.org/, last visited May 2, 2007.

`org.xml.sax`. This means that this class implements all callback functions for the SAX parser. The most important callbacks are the following:

**startDocument** is called at the beginning of the XML document and is used for some initialisation.

**endDocument** is called after the last closing element. At this point the XTAEMS parser starts resolving references between objects.

**startElement** is called every time a start element is found. This callback provides the name and attributes of the element. At this point the XTAEMS parser constructs a TÆMS object corresponding to the element's name and sets all information available through the attributes. Then the object is put on the parse stack and registered in an internal object registry. Not all tags have corresponding TÆMS objects. In that case information provided by these tags can be added directly to the object on the top of the stack.

**endElement** is called every time a closing tag is found. This callback provides only the element's name. At this point the parser knows that all information for the corresponding TÆMS object has been read.

The parser's main functional units are `parseStack`, `registry`, `taskOrMethodReferenceRegistry`, `taskOrMethodReferencePool`, `agentReferencePool`, and `resourceReferencePool`.

`parseStack` keeps track of objects to be refined by child elements in the XML description. Figure 2.2 illustrates this idea. `registry` is a hashtable containing references to all XTAEMS objects having an `id` attribute serving as the key. `taskOrMethodReferenceRegistry` stores all subtask relationships. `taskOrMethodReferencePool` is a set containing the `id`s of all tasks or methods which have been referenced. `agentReferencePool` is a set containing the `id`s of all agents which have been referenced. `resourceReferencePool` is a set containing the `id`s of all resources which have been referenced. The main purpose of the latter three entities is detection of referenced but never defined tasks, agents, and resources.

We have added a method `finishDefinition()` to all TÆMS objects. This method is called when the end element of the XML description for this object has been reached. At this point the object is removed from the stack. This method is the right place to carry

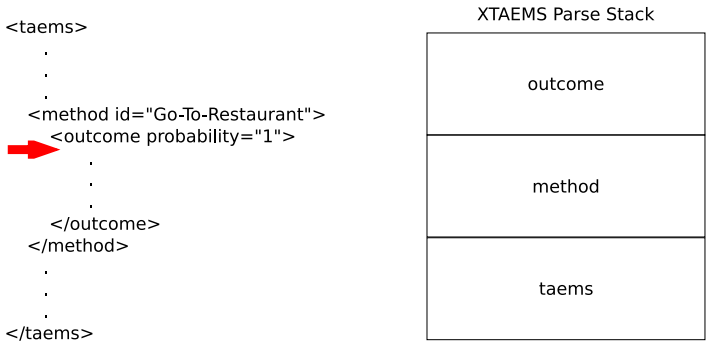


**Figure 2.2.:** Parser while parsing a XTAEMS structure. The red arrow indicates the position of the parser. The box to the right illustrates the objects on the parse stack.

out initial calculations performed by the constructor in the former version of VIE-CDS. This has to be deferred until this time because the bulk of information for a TÆMS object is unknown until its end tag has been reached.

When the `endDocument()` callback is called the parser starts resolving references between objects. This process iterates over all objects stored in the `registry` and performs the following steps:

1. If the object is a method, then check if it is also referenced and check if the referenced agent is also defined.
2. If the object is a task, check if the referenced agent is also defined. Then, iterate over all references in `taskOrMethodReferenceRegistry` and assign the appropriate objects, found in the `registry` via the `id`s, to this object. If this object is not referenced in `taskOrMethodReferencePool` assume it is a `task_group`.
3. If the object is a resource, check the `resourceReferencePool` if it also gets referenced. Then check if the referenced agent is also defined.
4. If the object is an interrelationship, check if it interrelates only objects which exist and can be interrelated by an interrelationship of this type. Check if only defined agents are referenced.
5. If the object is an agent definition, check if this agent is also referenced.

Upon successful completion of all steps, a valid in-memory TAEMS structure has been constructed.

### 2.3.1. Parser Analysis

As specified in the list of design goals at the beginning of this chapter, processing speed was included as a criterion, so as to provide a tool of practical relevance. In concrete terms, we wanted the XTAEMS parser to be at least as fast as the previous TTAEMS parser. Here, we present a short performance comparison of the TTAEMS and XTAEMS parsers.

For parser analysis, we have built a tool that creates random heterarchical XTAEMS structures with a fixed number of tasks, methods, resources, and interrelationships. Then it creates a TTAEMS structure via the XSLT stylesheet. After that, the time from the call of the parser's `parse()` method until it returns an in-memory TÆMS structure is clocked. This clocking is done for TTAEMS and the corresponding XTAEMS encoding. The following information into a tab-separated text file:

**date:** timestamp of the entry

**run-time:** time elapsed between the call of `parse()` and its return with an in-memory TÆMS structure.

**type:** the parser used. Possible values are XTAEMS or TTAEMS

**tasks:** number of tasks

**methods:** number of methods

**cResources:** number of consumable resources

**ncResources:** number of non consumable resources

**produces:** number of produces interrelationships

**consumes:** number of consumes interrelationships

**interrelations:** number of interrelationships consisting of enables, disables, facilitates, and hinders which are equally distributed in the structure

**avgChildren:** the average number of subtasks a task has

**minChildren:** the minnimum number of subtasks a task must have

**maxChildren:** the maximum number of subtasks a task may have

**filesize:** number of characters in the file (including whitespaces)

```
d$type        2 2.6192e+10 1.3096e+10   21118 < 2.2e-16 ***
Residuals 1824 1.1311e+09 6.2012e+05
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

**Listing 2.5:** ANOVA table for run-times without outlier values.

We used R[11] to carry out statistical analysis. Figure 2.3 gives an overview of how the number of nodes in a TÆMS structure is interrelated with the run-times of the used parser. We define a node as everything in TÆMS which has an `id` in XTAEMS or a `label` field in TTAEMS. Triangles represent TTAEMS run-times and circles XTAEMS run-times. Run-time is measured in micro seconds. Two things catch our eyes. The first is, the significant number of outliers for both parsers. Even though we could not determine a correlation of our measured attributes and the outliers. Our hypothesis is that the larger the number of nodes, the more susceptible the run-time gets to scheduling issues of the underlying operating system. It might be the case that the operating system scheduler suspends the Java Virtual Machine more often and therefore the run-time increases. For the time being, we consider all instances with a run-time above 5000 $\mu s$ as outliers and remove it from our data set. After this removal the plot looks as illustrated in figure 2.4. It shows that XTAEMS run-times spread more than TTAEMS run-times, but there is definitely a large amount of runs where the XTAEMS parser is faster than the TTAEMS parser. This observation gets confirmed by the boxplot in figure 2.5. The median of run-times of the XTAEMS parser is definitely smaller than the median of run-times of the TTAEMS parser. The whiskers in the plot also confirm that the XTAEMS parser run-times spread more than the TTAEMS parser run-times. Additionally, we have performed an ANOVA to determine wheter the used parser is an important factor. Listing 2.5 shows R's ANOVA table. Even if we do not remove the outliers the picture is the same as indicated by R's ANOVA table shown in listing 2.6.

The runtime differences of TTAEMS and XTAEMS parsers are statistically relevant. In conjunction with the information from the plots we conclude that the XTAEMS parser is faster than the TTAEMS parser and we therefore reached our goal to at least match the performance of the TTAEMS parser.

Apart from the run-time comparison, another interesting aspect would be to estimate the

[11] http://www.r-project.org/, last visited May 2, 2007.

```
             Df      Sum Sq     Mean Sq F value     Pr(>F)
d$type        2 4.9769e+11 2.4884e+11   538.45 < 2.2e-16 ***
Residuals  4318 1.9955e+12 4.6215e+08
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

**Listing 2.6:** ANOVA table for run-times containing outlier values.

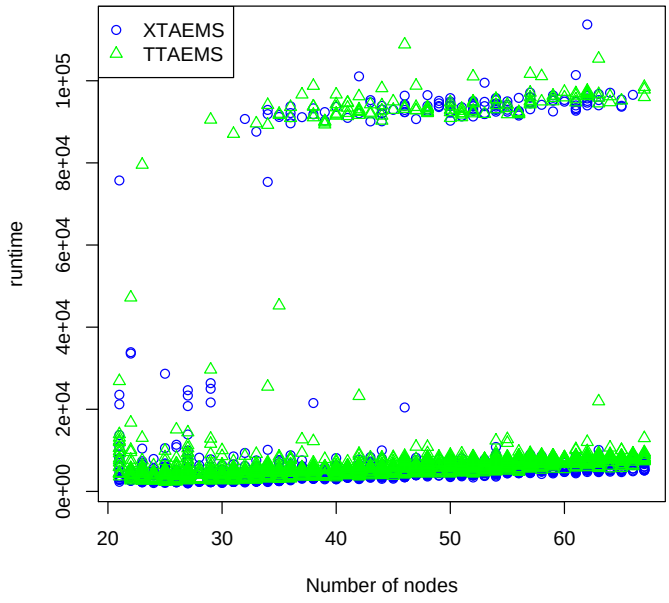


**Figure 2.3.:** Comparison of the run-time behaviour of the XTAEMS and the TTAEMS parsers in relation to the number of nodes in the TÆMS structure definition file. The run-time is given in micro-seconds.

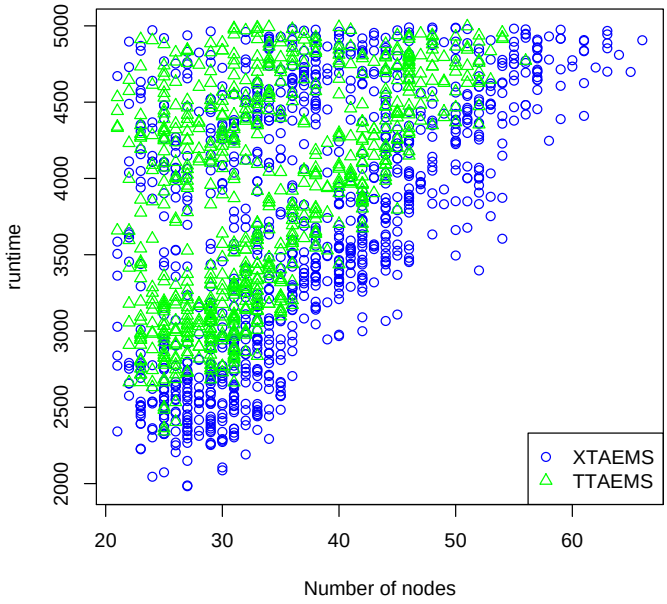


**Figure 2.4.:** Comparison of the run-time behaviour of the XTAEMS and the TTAEMS parsers in relation to the number of nodes in the TÆMS structure definition file without outlier values. The run-time is given in micro-seconds.

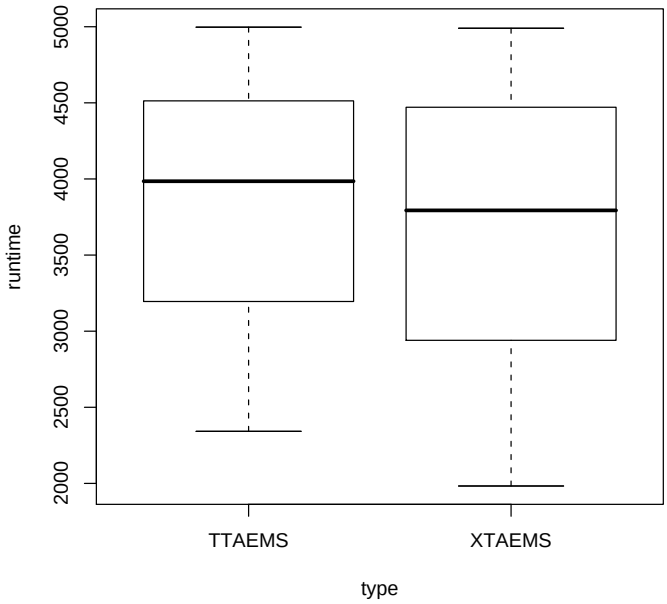


**Figure 2.5.:** Comparison of run-time behaviour in a boxplot. The run-time is given in micro-seconds.

number of nodes in a TÆMS structure from file size. If this was possible it would allow further optimisations in our XTAEMS parser and maybe also on the TTAEMS parser: If the number of nodes could be estimated in advance, the size the hashtables could be adjusted avoiding the need to rehash, with related performance enhancements. Figure 2.6 shows a plot of the number of nodes in relation to the file size. Circles represent XTAEMS files and triangles TTAEMS files. The dashed lines are the two linear models estimating the number of nodes. We can see that both estimations are quite good, even though the bigger XTAEMS files spread more. The Quantile-Quantile plots in figure 2.7 and 2.8 prove that the residuals of both data sets are normally distributed. Therefore it is in fact possible to predict the number of nodes from the file size, which opens the door for further optimisations.

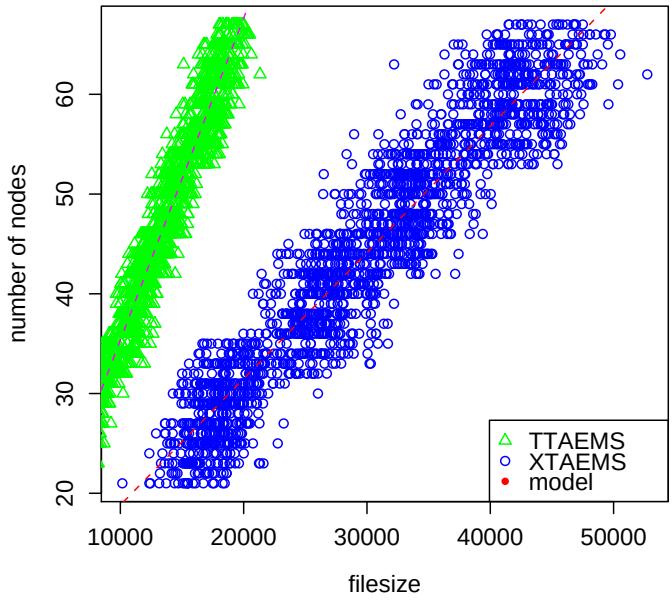


**Figure 2.6.:** Number of nodes in relation to the file size. The file size is given in bytes.

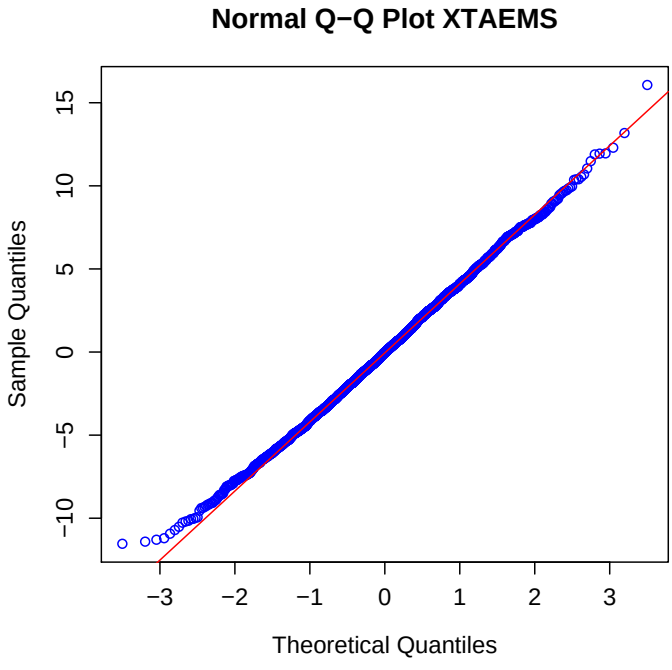


**Figure 2.7.:** Quantile-Quantile plot for the linear model of XTAEMS

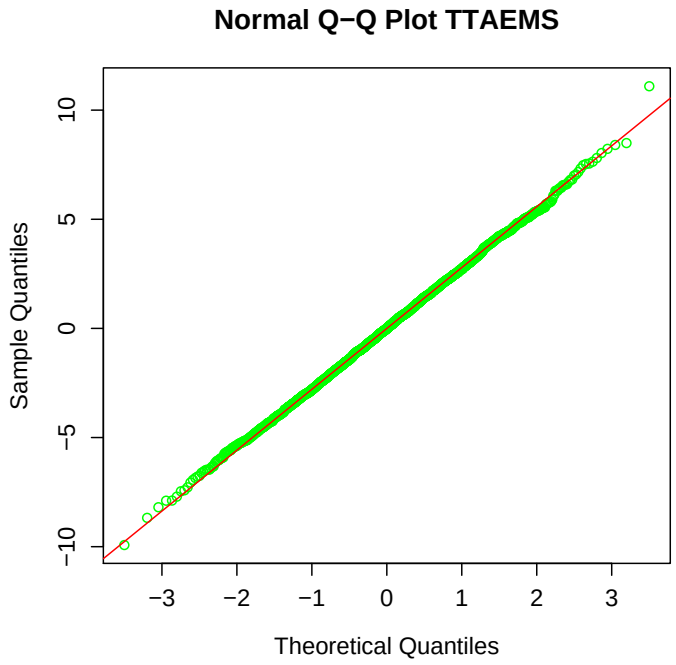


**Figure 2.8.:** Quantile-Quantile plot for the linear model of TTAEMS

# 3. Design-to-Criteria Scheduling Integration

JADE is the multi-agent platform of choice for our reference implementation. As mentioned in the introduction, JADE is a FIPA compliant multi-agent framework. Accordingly, our aim was also to design the integration of a scheduling component in as generic a way as possible, so as to ideally enable all FIPA compliant agent frameworks to adopt our approach and thereby even enable inter-platform scheduling.

Section 3.1 introduces the generic design common for all FIPA compliant platforms. In section 3.2 we discuss our implementation of these generic concepts in JADE.

## 3.1. Conceptual Design

In the design phase, numerous trade-offs between flexibility and usability had be taken. In particular, we do not aim to answer the question wheter it is better to use dedicated planning, scheduling and execution monitoring agents or to provide these abilties to each agent. Therefore, we rather speak about agent roles than concrete agents in the following. Interaction protocols are the glue between these roles.

One major design decision is to treat planning, scheduling and execution monitoring as one big building block. Consequently, these tasks cannot be distributed among different agents. This approach restricts the multi-agent system designer, but saves communication costs. It avoids sending large string representations of TÆMS structures multiple times over the communication channel.

With our approach it is possible to build *centralised* (only one agent plays the planning, scheduling, and execution monitoring role), *multi-centric* (several agents have planning,

scheduling, and execution monitoring abilities) as well as *fully distributed* (resembling GPGP: each agent has planning, scheduling, and execution monitoring abilites) infrastructures. This is our contribution to flexiblity. For usability, we require each agent to offer a mechanism for manipulating its local TÆMS structures. This mechanism has to take care of properly proclaiming local TÆMS knowledge (see section 3.1.2), as well as implementing all scheduling related interaction protocols (see sections 3.1.1 and 3.1.3). To summarise, we do not want the agent programmer to be confronted with internals of the scheduling and method execution process, but still provide the greatest possible degree of control.

### 3.1.1. Ontologies

According to the FIPA specifications each content language token has to have a certain ontology type. This is necessary to define a content language's formal semantics. Here, we relate to the semantics of SL. An ontology item must then have one of the following types [Bellifemine et al., 2007].

**Predicates** say something about the status of the world an may evaluate to `true` or `false`.

**Terms** are expressions identifying entities (abstract or concrete) that "exist" in the world. Agents may reason about them. They are further classified into:

**Primitives** are basic types like string, integer, and float.

**Concepts** are structured entities. They can be best thought of as "things" which have names and certain characteristics described by so-called *slots*. Concepts do not make sense as sole content of an ACL message.

**Agent actions** are special concepts. They indicate the actions that can be performed by an agent.

**Aggregates** are groups of entities, such as sequences or sets.

**Identifying referential expressions** identify entities for which a given predicate is `true`.

We refer to an ontology as a group of semantically coherent *predicates*, and *terms*. It is possible to extend or refine an existing ontology by another, resembling the inheritance mechanism of object oriented modelling. We have developed three such ontologies to provide the vocabulary necessary for integrating scheduled task execution in an FIPA compliant multi-agent architecture. Figure 3.1 gives an overview of the relationships between these ontologies which we discuss in the following in greater depth.

Based on ideas taken from [Cranefield and Purvis, 1999], we use the well-known UML formalism to visualise ontology structures: *Packages* refer to an ontology in our sense. *Arrows* with hollow heads define generalisation relationships. *Dashed arrows* indicate dependency relationships. *Solid lines* define associations. A hollow *diamond* indicates an aggregation (this means that an entity mainly consists of another entity). The UML *class* symbol is used to describe an entity. *Stereotypes* are used to indicate entity types. The *attributes* field is used to describe the slots of a concept. The *plus sign* marks a slot value as mandatory, whereas a *minus sign* declares a slot value to be optional.

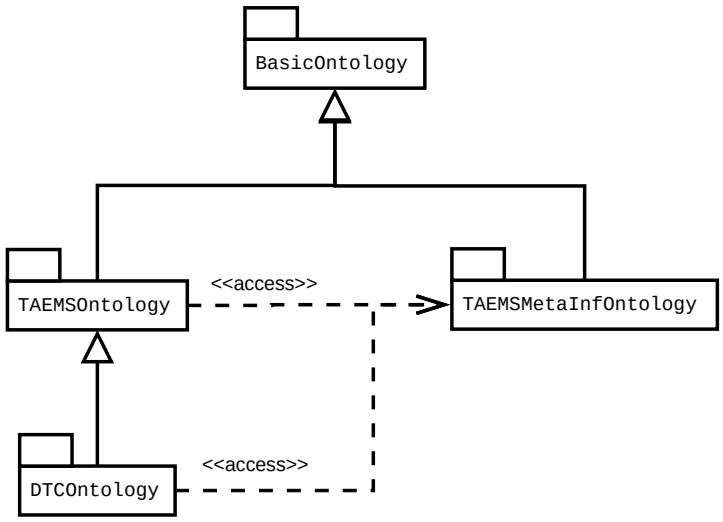


**Figure 3.1.:** Relationships between *TAEMSOntology*, *DTCOntology* and *TAEMSMetaInfOntology*.

**BasicOntology** is the root ontology we extend. We assume it to contain all primitive concept and aggregate definitions. Inspired by JADE's ontology mechanism, we expect the multi-agent infrastructure to provide us with this ontology. Our on-

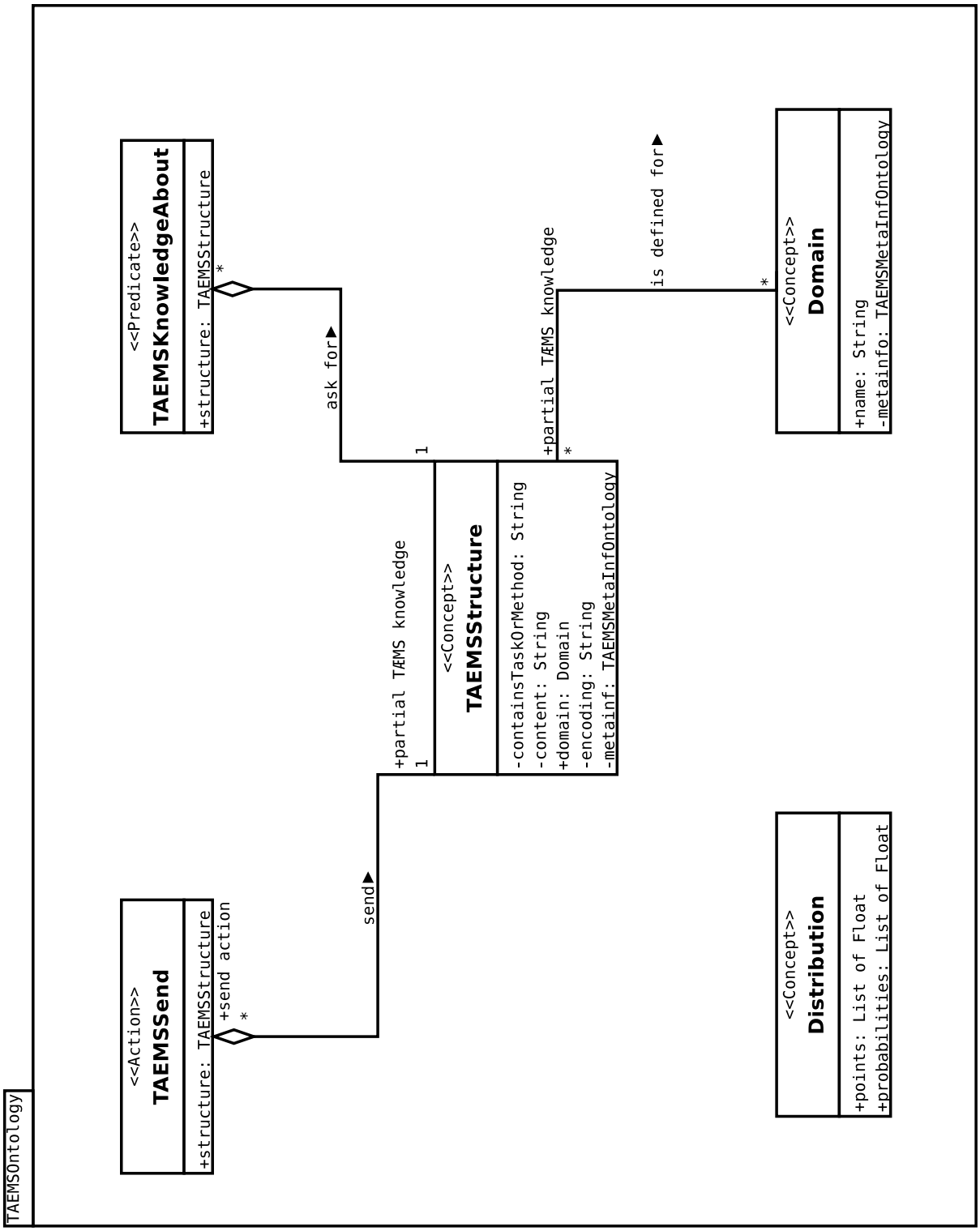


**Figure 3.2.:** *Actions*, *concepts*, and *predicates* contained in *TAEMSOntology*.

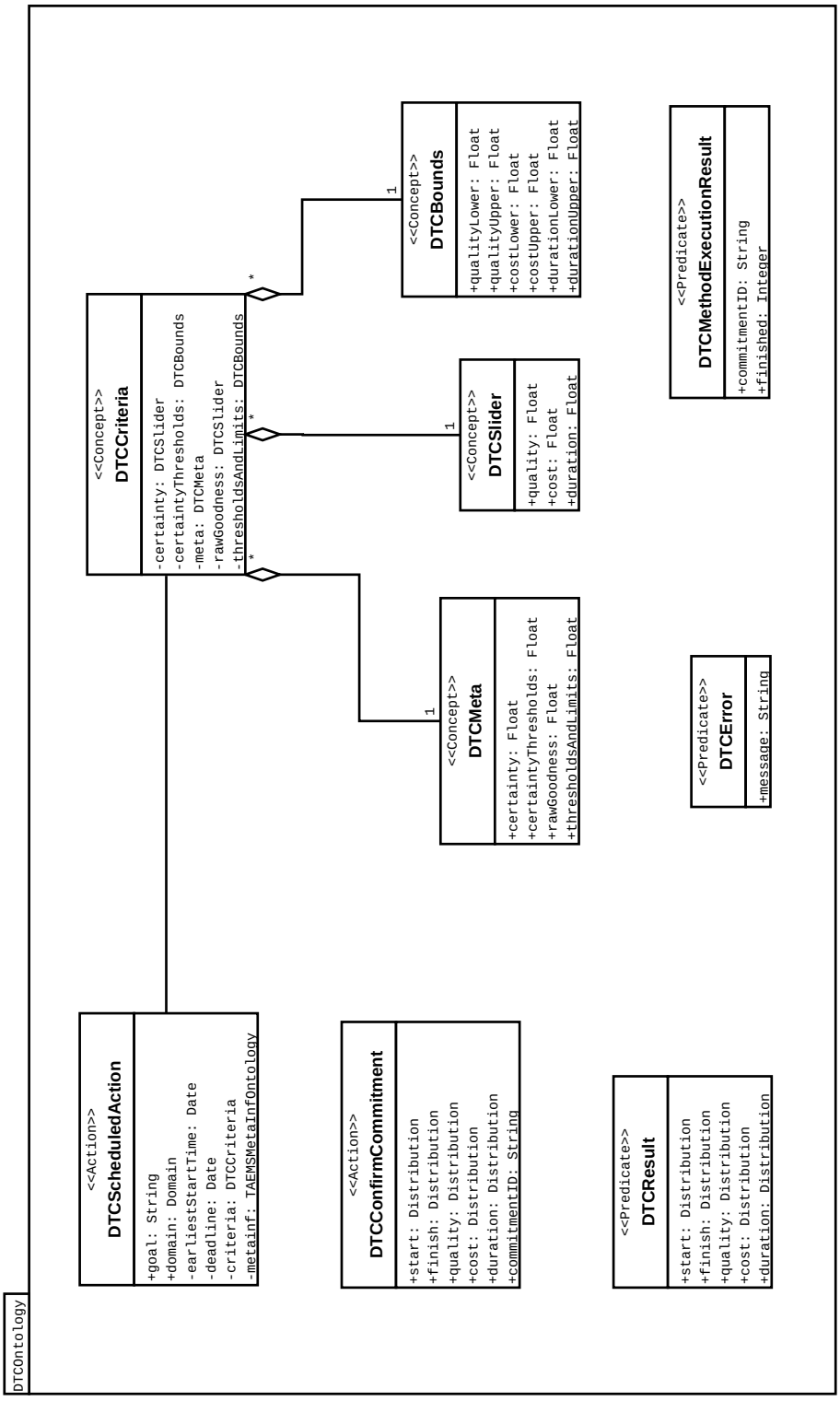


**Figure 3.3.:** *Actions*, *concepts*, and *predicates* contained in *DTCOntology*.

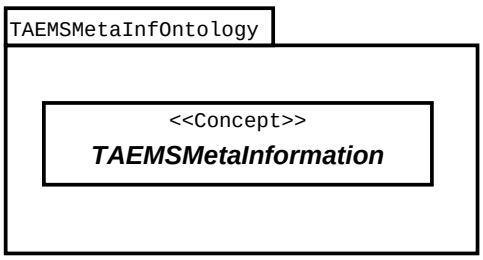


**Figure 3.4.:** The *TAEMSMetaInfOntology* contains only one *concept*. It is designed to be refined with application specific meta-information.

tologies require the three primitive types *integer*, *float*, and *string*, as well as the aggregate type *sequence*.

**TAEMSOntology** mainly deals with wrapping TÆMS structures, defining the vocabulary for TÆMS structure exchange and providing a *predicate* to refine a given TÆMS structure. Figure 3.2 provides an overview of this ontology's structure.

**TAEMSStructure** is the heart of this ontology. This concept wraps TÆMS structures, allowing to use them in content languages. The slot `containsTaskOrMethod` contains the name of a TÆMS task or method. The slot `content` contains a string representation of a TÆMS structure encoded as defined by the slot `encoding`. This structure must contain the TÆMS task or method referenced in the slot `containsTaskOrMethod`. The slot `metainf` allows the specification of further meta-information about the wrapped TÆMS structure.

**TAEMSSend** is the action used to exchange TÆMS structures. The slot `structure` contains the *TAEMSStructure* to be sent.

**TAEMSKnowledgeAbout** is a predicate that wraps the concept *TAEMSStructure*. The special thing about this predicate is that it expects a TÆMS method definition with all execution qualities and nothing else, whereas in *TAEMSSend* any other TÆMS information can be included the sending agent considers important.

**TAEMSDomain** is a concept that defines a field of knowledge. Each TÆMS structure is assigned to such a domain.

**Distribution** is a concept that extends *BasicOntology*. It offers the possiblity to use the discrete probability distributions omnipresent in TÆMS.

**DTCOntology** adds DTC-specific predicates and terms to *TAEMSOntology*. Its main purpose is to define the vocabulary necessary for requesting scheduled method execution, as well as providing VIE-CDS with scheduling criteria. Figure 3.3 gives an overview of the structure of *DTCOntology*.

**DTCScheduledAction** is an action that allows requesting a DTC scheduled task execution. The slot `goal` must contain a TÆMS task or method from the TÆMS domain specified in the slot `domain`. It is possible to specify an earliest start time and a deadline for execution in the slots `earliestStartTime` and `deadline` as well as additional meta-information in the slot `metainf`. The slot `criteria` allows the definition of scheduling criteria.

**DTCCriteria** is a concept that aggregates the scheduling criteria for a certain scheduling request following the *slider metaphor*. For the meaning of these values, please refer to [Wagner et al., 1998, Jung, 2003]. If no scheduling criteria are specified the default ones of the scheduler are used.

**DTCMeta** is a concept that wraps values for the relative importance of *raw goodness*, *thresholds and limits*, *certainty*, and *certainty thresholds*.

**DTCSlider** is a concept that defines a value for *quality*, *cost*, and *duration*.

**DTCBounds** is a concept that defines an upper and a lower bound for *quality*, *cost* and *duration*.

**DTCConfirmCommitment** is an action that request, an agent to confirm a commitment to a certain TÆMS method execution. The slots `start` and `finish` define the possible start and finish time distributions calculated by the scheduler. The slots `quality`, `cost` and `duration` describe the expected qualities of TÆMS method execution. The slot `commitmentID` contains an ID identifying this commitment.

**DTCResult** is a predicate that describes the expected or real characteristics of schedule execution. This includes start time of execution, finish time, as well as the expected schedule execution qualities.

**DTCMethodExecutionResult** is a predicate that indicates that the method defined by the commitment ID in the slot `commitmentID` has been successfully executed and finished at the time provided in the slot `finished`.

**DTCError** is a predicate used to describe an error which occured during scheduling.

**TAEMSMetaInfOntology** contains only one concept called *TAEMSMetaInformation*, as illustrated in figure 3.4. This ontology is intended to be refined by application specific ontologies which provide meta-information about information encoded in a TÆMS structure. For example, this ontology could be used to provide information about how to resolve redundancies.

### 3.1.2. Publish and Discover

FIPA compliant multi-agent infrastructures must provide a yellow pages service called *directory facilitator* (DF). On the one hand, we make standard use of it to register the planning, scheduling, execution monitoring service, and on the other hand we also exploit it to publish an agent's local TÆMS knowledge. To avoid the DF to be a single point of failure, we recommend to use techniques such as DF *replication* and *federation*.

A DF can administer multiple *service descriptions* per agent. Each service description must have a *name* and a *type*. It may have multiple additional properties specified as key-value pairs, as well as a set of known ontologies and interaction protocols.

#### TÆMS Knowledge

We expect each agent owning local TÆMS structures to announce the contained TÆMS tasks and methods with the DF. We expect the agent to publish only methods which it also can execute. To be able to distinguish standard service definitions from TÆMS information, we introduce the new service type `taems`. For each known TÆMS domain, the agent must register a service of type `taems`. The service name must be equal to the domain name. Each unique type-name pair contains a set of properties. The property's value may be `task` or `method`. Its name corresponds to a TÆMS tasks or method name. Another special property with name `encoding` exists. It indicates the agent's preferred TÆMS encoding. This feature allows the mapping of TÆMS tasks or methods to FIPA compliant agents using standard FIPA DF mechanisms.

Listing 3.1 shows the content of an ACL message which asks the DF for TÆMS method `Treatment1` in domain `ClinicalCentre` with `xtaems` as the preferred encoding.

```
((action
     (agent-identifier
       :name df@arwen:1099/JADE
       :addresses (sequence http://arwen:7778/acc))
     (search
       (df-agent-description
         :services
           (set
             (service-description
               :name ClinicalCentre
               :type taems
               :properties
                 (set
                   (property
                     :name Treatment1
                     :value method)
                   (property
                     :name encoding
                     :value xtaems)))))
       (search-constraints
         :max-results -1))))
```

**Listing 3.1:** Request (in FIPA-SL) from the yellow pages service all agents having TÆMS knowledge about TÆMS method *Treatment1* in domain *ClinicalCentre* which support encoding TÆMS structures in XTAEMS.

```
((action (agent-identifier
       :name df@arwen:1099/JADE
       :addresses (sequence http://arwen:7778/acc))
     (search
       (df-agent-description
         :services
           (set
             (service-description
               :name DTCScheduling
               :type TAEMSPlanScheduleMonitor
               :protocols (set DTCSchedule)
               :ontologies (set DTCOntology))))
       (search-constraints
         :max-results -1))))
```

**Listing 3.2:** Request (in FIPA-SL) from the yellow pages service all agents providing a design-to-criteria scheduling service and speaking the `DTCSchedule` protocol.

#### Design-to-Criteria Scheduling Service

An agent having DTC planning, scheduling and execution monitoring abilities has to register this service with the DF as service of type `TAEMSPlanScheduleMonitor` and name `DTCScheduling`. It also must declare to speak the `DTCSchedule` interaction protocol and knows the `DTCOntology`.

Listing 3.2 shows the content of an ACL message querying the DF for a plann, scheduling, and execution monitoring service.

### 3.1.3. Scheduling Phases

Our scheduling integration approach does not comprise a single component to be added to a FIPA compliant agent platform. We specify it as an orchestration of well defined FIPA interaction protocols, namely *FIPA Request Interaction Protocol* (see figure 3.5) and *FIPA Query Interaction Protocol* (see figure 3.6), as well as *FIPA Cancel Meta-Protocol* (see figure 3.7). We also make use of the slight modification proposed by the JADE community, to make the `AGREE` message optional if the `INFORM` message is immediately available.

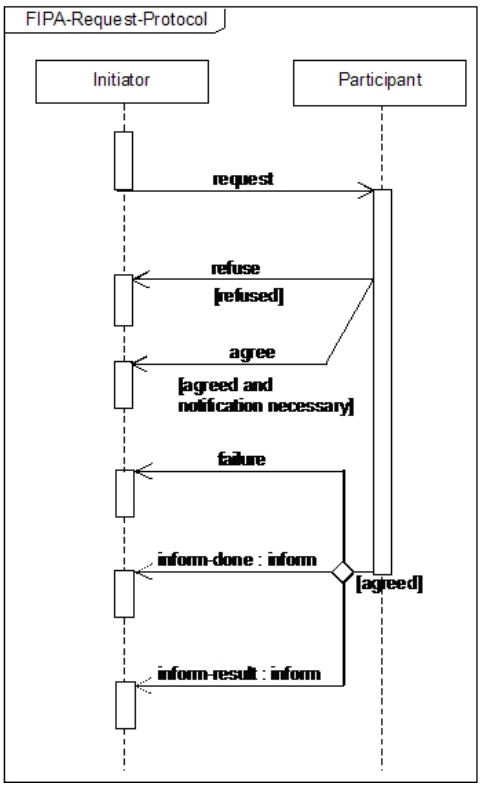


**Figure 3.5.:** FIPA Request Interaction Protocol,
[Foundation for Intelligent Physical Agents (FIPA), 2002b, p.1]

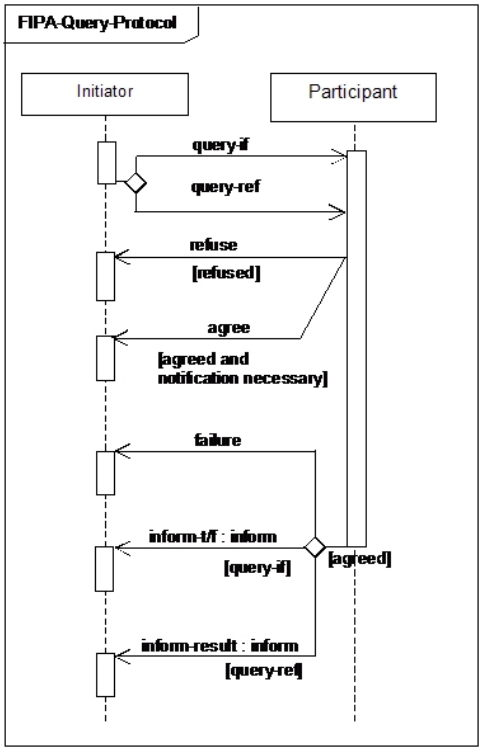


**Figure 3.6.:** FIPA Query Interaction Protocol,
[Foundation for Intelligent Physical Agents (FIPA), 2002c, p.1]

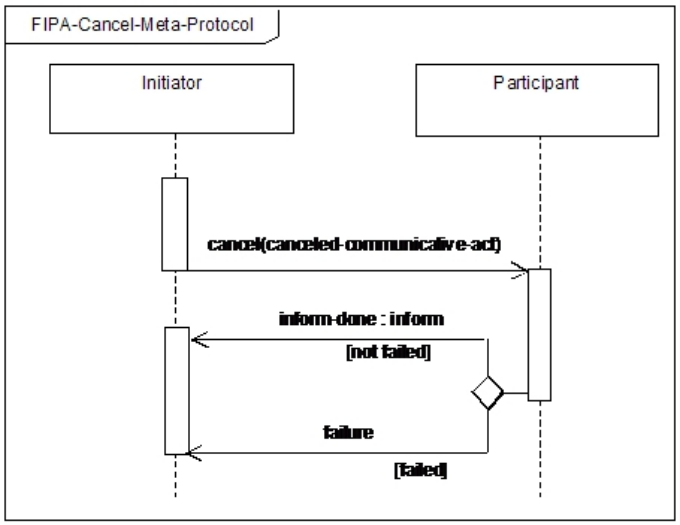


**Figure 3.7.:** FIPA Cancel Meta-Protocol,
[Foundation for Intelligent Physical Agents (FIPA), 2002b, p.2]

Figure 3.8 provides an overview of the protocol flow of our approach. We use an UML-like formalism as proposed in [Odell et al., 2001]: *Requester*, *Plan/Schedule/Monitor execution* and *TAEMS Provider* do again refer to roles an agent can play during scheduled action execution. The duplicate occurrence of TAEMS Provider with dots in-between should illustrate that multiple TAEMSProvider roles could be involved during one scheduled action execution.

The whole scheduling process is triggered by a schedule request. If the planning, scheduling, and execution monitoring role accepts to coordinate the scheduling process, it runs through three phases delineated by the first three grey rectangles on the right. After that it can inform the requester about the expected scheduling qualities and execution time. According to the start times calculated by the scheduler each agent involved in scheduled action execution starts self-dependently TÆMS method execution and indicates its result to the plan, schedule, execution monitoring role.

When all TÆMS methods have been executed successfully, the requester is informed about the finished schedule execution and the final qualities achieved.

Each scheduling phase has been given a protocol name: *DTCSchedule*, *TAEMSSend*, *TAEMSKnowledge*, and *DTCMethodExecution*. We do not propose new interaction protocols, rather, this approach allows filtering out messages by the scheduling component, freeing the agent programmer from the responsibility of *not* handling these messages. We now discuss the scheduling phases in more detail.

#### Request Scheduled Action Execution

The request for scheduled action execution triggers the whole planning, scheduling and action execution monitoring process. It is implemented as FIPA Request Interaction Protocol. We call this protocol *DTCSchedule*. The requester queries the DF for an agent providing a planning, scheduling, and execution monitoring service as shown in listing 3.2. Then it requests a scheduled action execution with a message like the one given in listing 3.3. A `REQUEST` message in protocol DTCSchedule is expected to

- contain a *DTCScheduledAction* action,
- indicate that it uses *DTCOntology*,

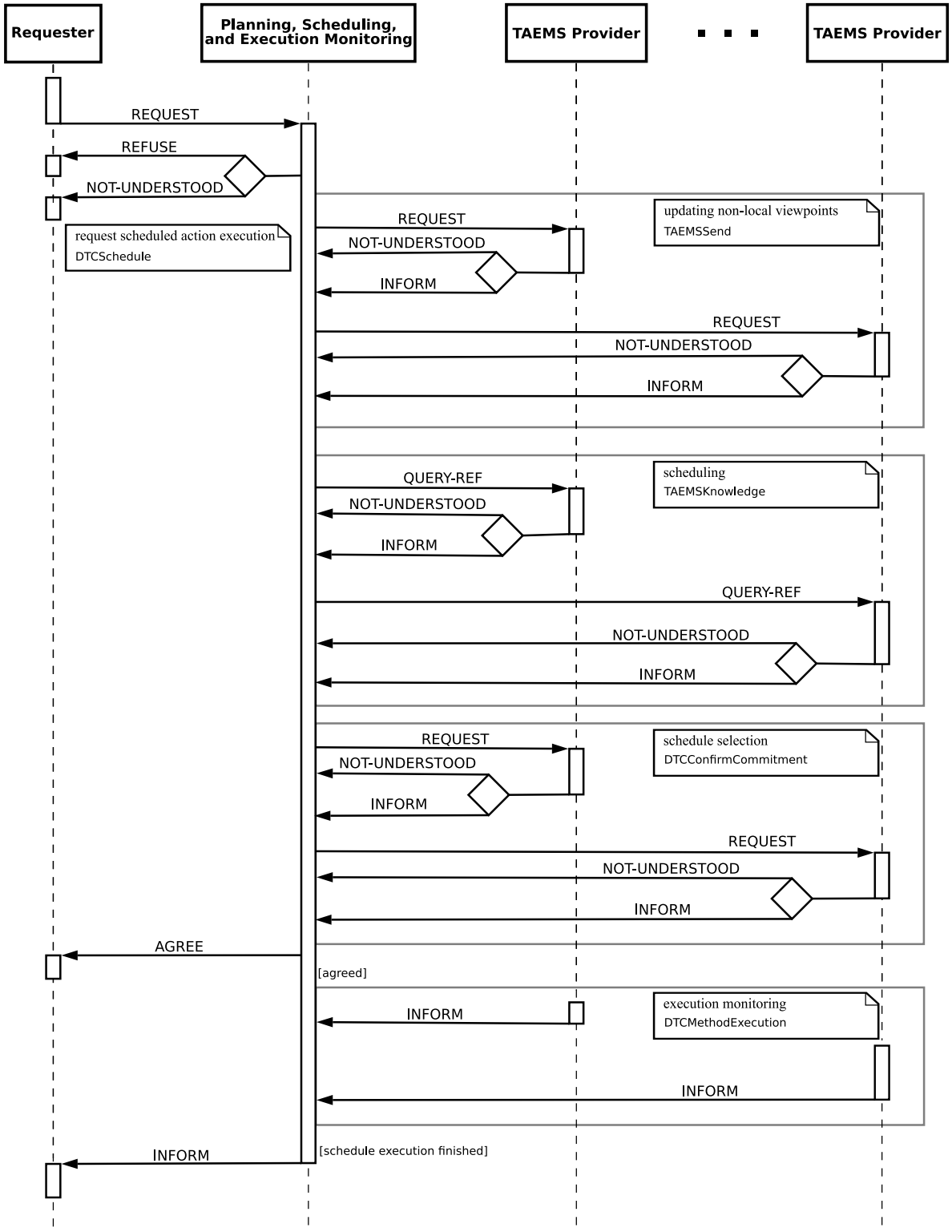


**Figure 3.8.:** Protocol flow for scheduled action execution

```
(REQUEST
:sender  (agent-identifier
  :name Requester@arwen:1099/JADE
  :addresses (sequence http://arwen:7778/acc )
  :X-JADE-agent-classname ofai.jade.examples.hospital.Requester )
:receiver  (set
 (agent-identifier
  :name Planner1@arwen:1099/JADE
  :addresses (sequence http://arwen:7778/acc )))
:content  "(
 (action
  (agent-identifier
    :name Planner1@arwen:1099/JADE
    :addresses (sequence http://arwen:7778/acc))
    (DTCScheduledAction
      :criteria (DTCCriteria
          :certainty (DTCSliders :quality 50.0 :cost 50.0 :duration
             50.0)
        :certaintyThresholds (DTCBounds :qualityLower 0.0 :qualityUpper
            50.0
        :costLower 0.0 :costUpper 50.0 :durationLower 0.0 :
           durationUpper 50.0)
        :meta (DTCMetaSliders :certainty 50.0 :certaintyThresholds 50.0
            :rawGoodness 50.0 :thresholdsAndLimits 50.0)
        :rawGoodness (DTCSliders :quality 50.0 :cost 50.0 :duration
           50.0)
          :thresholdsAndLimits (DTCBounds :qualityLower 0.0 :
             qualityUpper 50.0 :costLower 0.0 :costUpper 50.0 :
             durationLower 0.0 :durationUpper 50.0))
        :domain (TAEMSDomain :name ClinicalCentre)
        :goal FullTreatment)))"
:language fipa-sl
:ontology DTCOntology
:protocol DTCSchedule
:conversation-id  ScheduleID26790717117507865461 8 )
```

**Listing 3.3:** A sample scheduling request message.

```
(REQUEST
:sender  (agent-identifier
  :name Planner1@arwen:1099/JADE
  :addresses (sequence http://arwen:7778/acc )
  :X-JADE-agent-classname ofai.jade.examples.planner.PlannerAgent)
:receiver  (set
  (agent-identifier
    :name Orthopaedics@arwen:1099/JADE
    :addresses (sequence http://arwen:7778/acc)))
:content  "((action
  (agent-identifier
    :name Planner1@arwen:1099/JADE
    :addresses (sequence http://arwen:7778/acc))
    (TAEMSSend
      :structure (TAEMSStructure
        :containsTaskOrMethod FullKneeCheck
        :domain (TAEMSDomain :name ClinicalCentre)
        :encoding xtaems))))"
:reply-with  R1175078656775_0
:language  fipa-sl
:ontology  TAEMSOntology
:protocol  TAEMSSend
:conversation-id  C446196_1175078656775 )
```

**Listing 3.4:** A TÆMS knowledge request.

- indicate *DTCSchedule* as protocol, and
- have a *unique conversation-ID* which will be used to reference this scheduling request.

If one of the conditions above is not met, the planning, scheduling, and execution monitoring agent answers with a `NOT-UNDERSTOOD` message. If all conditions are met, the planning, scheduling, and execution monitoring agent can `REFUSE` the request, or it continues with the first three phases of scheduled method execution. These phases form the core of design-to-criteria scheduling integration.

```
:sender  (agent-identifier
  :name Orthopaedics@arwen:1099/JADE
  :addresses (sequence http://arwen:7778/acc)
  :X-JADE-agent-classname ofai.jade.examples.hospital.StationAgent)
:receiver  (set
  (agent-identifier
    :name Planner1@arwen:1099/JADE
    :addresses (sequence http://arwen:7778/acc )
    :X-JADE-agent-classname ofai.jade.examples.p lanner.PlannerAgent ))
:content  "((action
  (agent-identifier
    :name Orthopaedics@arwen:1099/JADE
    :addresses (sequence http://arwen:7778/acc))
    (TAEMSSend
      :structure (TAEMSStructure
      :containsTaskOrMethod FullKneeCheck
      :content \"<?xml version=\\"1.0\\"
encoding=\\"UTF-8\\"?> <taems>
    <task id=\\"FullCnemialTreatment\\" qaf=\\"sum_all\\">
        <subtask ref=\\"doCnemialTreatment\\"/>
    </task>
    <task id=\\"doCnemialTreatment\\" qaf=\\"max\\">
        <subtask ref=\\"Cnemial2\\"/>
    </task>
    <method id=\\"Cnemial1\\"/>
</taems>\"
      :domain (TAEMSDomain :name ClinicalCentre)
      :encoding xtaems))))"
:reply-with  Planner1@arwen:1099/JADE1175078656846
:in-reply-to  R1175078656775_0
:language  fipa-sl
:ontology  TAEMSOntology
:protocol  TAEMSSend
:conversation-id  C446196_1175078656775 )
```

**Listing 3.5:** A response to the TÆMS knowledge request specified in listing 3.4.

### Updating Non-Local Viewpoints

In the majority of cases, the planning, scheduling, and execution monitoring agent does not have enough local TÆMS knowledge to build a TÆMS structure achieving the desired goal. Therefore, it updates its non-local viewpoints and builds a *partial global TÆMS structure* which we also refer to as partial global plan, even though we know the term multi-plan would be more appropriate. This updating occurs as the planning, scheduling, and execution monitoring agent looks at its already built partial global plan and queries DF (see listing 3.1) for all tasks and methods in this TÆMS structure for agents having TÆMS knowledge about these tasks or methods. Afterwards, it requests these agents to send TÆMS knowledge they consider important for the requested task or method. Requesting is done with the FIPA Request Interaction Protocol, omitting the intermediate agree step. We call this protocol *TAEMSSend*. Listing 3.4 shows a sample request for partial TÆMS structures and listing 3.5 gives the corresponding reply. The received message is expected to contain the predicate *TAEMSKnowledgeAbout* and use the ontology *TAEMSOntology*. In this phase, agents are not expected to send quality, cost, or duration information of methods. This step only serves exploring the problem structure. Then, the receiver agent merges the received TÆMS structures into the existing stub. It continues with this until no tasks or methods arrive which could be refined further. During this merging operation, the merging component has to take care that no unnecessary tasks or methods are refined. If the requester specifies an earliest start time or a deadline, these values are set at the goal node of the TÆMS structure as attributes `earliestStartTime` and `deadline` (in case of XTAEMS encoding).

The following guidelines guarantee a goal directed task and method refinement:

- Treat the whole TÆMS structure as a tree with the scheduling goal as root of it;
- Delete all supertasks of the scheduling goal;
- Treat *enables*, *disables*, *facilitates*, and *hinders* as a parent-child interrelationships with the `to`-node as parent and the `from`-node as child;
- Treat all resources as nodes in the goal tree;
- Treat all *consumes* and *produces* as parent-child interrelationships with the `from`-node (method) as parent and the `to`-node (resource) as child;

```
(QUERY-REF
:sender  (agent-identifier
  :name Planner1@arwen:1099/JADE
  :addresses (sequence http://arwen:7778/acc )
  :X-JADE-agent-classname ofai.jade.examples.planner. PlannerAgent )
:receiver  (set
 (agent-identifier
  :name Treatment1@arwen:1099/JADE
  :addresses (sequence http://arwen:7778/acc )))
:content  "((TAEMSKnowledgeAbout
 (TAEMSStructure
  :containsTaskOrMethod Treatment1
  :domain (TAEMSDomain :name ClinicalCentre))))"
:reply-with  R1175078657082_0
:language  fipa-sl
:ontology  TAEMSOntology
:protocol  TAEMSKnowledge
:conversation-id  C16586768_1175078657082 )
```

**Listing 3.6:** Request for a method execution commitment proposal.

- Remove all nodes and interrelationships not used in the tree.

Goal directed task and method refinement does not imply the existence of only a single top-level tasks. It is still possible that multiple top-level tasks are contained in the TÆMS structure, but all except the goal task are connected with the main goal tree through *enables*, *disables*, *facilitates*, or *hinders* interrelationships. We assume that the scheduler used on these structures groups the top-level tasks under a meta top-level task, using the quality accumulation function `sum_all`.

### Scheduling

The resulting partial global plan is passed to the scheduling part of the planning, scheduling, and execution monitoring component. If the scheduling request message contains scheduling criteria, the DTC scheduler is configured with them. Since so far only the problem structure has been determined it is now time to fetch proposals for concrete TÆMS method execution. Here, we use the FIPA Query Interaction Protocol omitting the intermediate agree step, and call it *TAEMSKnowledge*. The message content

```
:sender  (agent-identifier  :name Treatment1@arwen:1099/JADE
  :addresses (sequence http://arwen:7778/acc )
  :X-JADE-agent-classname ofai.jade.examples.hospit al.
     KneeTreatmentAgent )
:receiver  (set (agent-identifier
      :name Planner1@arwen:1099/JADE
      :addresses (sequence http://arwen:7778/acc )
      :X-JADE-agent-classname ofai.jade.examples.planner.PlannerAgent )
         )
:content  "((TAEMSKnowledgeAbout (TAEMSStructure :containsTaskOrMethod
   Treatment1
    :content \"<?xml version=\\"1.0\\" encoding=\\"UTF-8\\"?> <taems>
    <method id=\\"Treatment1\\">
        <outcome probability=\\"1.0\\">
            <quality>
                <distribution point=\\"1.0\\" probability=\\"1.0\\"/>
            </quality>
            <cost>
                <distribution point=\\"1.0\\" probability=\\"1.0\\"/>
            </cost>
            <duration>
                <distribution point=\\"1.0\\" probability=\\"1.0\\"/>
            </duration>
        </outcome>
    </method>
</taems>
\"
  :domain (TAEMSDomain :name ClinicalCentre)  :encoding xtaems)))"
:reply-with  Planner1@arwen:1099/JADE1175078657140  :in-reply-to
   R1175078657082_0
:language  fipa-sl  :ontology  TAEMSOntology  :protocol  TAEMSKnowledge
    :conversation-id  C16586768_1175078657082 )
```

**Listing 3.7:** A response to the method execution commitment proposal request given in listing 3.6.

```
(REQUEST
:sender  (agent-identifier :name Planner1@arwen:1099/JADE
  :addresses (sequence http://arwen:7778/acc )
  :X-JADE-agent-classname ofai.jade.examples.planner. PlannerAgent)
:receiver  (set ( agent-identifier :name Treatment1@arwen:1099/JADE ))
:content  "((action (agent-identifier :name Planner1@arwen:1099/JADE
  :addresses (sequence http://arwen:7778/acc))
  (DTCConfirmCommitment
    :start (Distribution :points (sequence 1.175078657E9) :
       probabilities
    (sequence 1.0))
    :finish (Distribution :points (sequence 1.175078658E9) :
       probabilities
    (sequence 1.0))
    :quality (Distribution :points (sequence 1.0) :probabilities (
       sequence 1.0))
    :cost (Distribution :points (sequence 1.0) :probabilities (sequence
        1.0))
    :duration (Distribution :points (sequence 1.0) :probabilities (
       sequence 1.0))
    :commitmentID C16586768_1175078657082)))"
:reply-with  R1175078657345_0  :language  fipa-sl
:ontology  DTCOntology  :protocol  DTCConfirmCommitment
:conversation-id  C14043096_1175078657345 )
```

**Listing 3.8:** A request for commitment confirmation.

is expected to be encoded in ontology *TAEMSOntology*. The used conversation-ID is very important because it is taken as identifier for this commitment proposal. Listings 3.6 and 3.7 show a request for commitment proposal and the corresponding reply.

Subsequently, the scheduler schedules this TÆMS structure and provides a set of possible schedules which is passed on to the next phase.

### Schedule Selection

The schedule selection phase is realised with the protocol *DTCConfirmCommitment*, which is based on the FIPA Request Interaction Protocol. The planning, scheduling, and execution monitoring component takes the a first schedule out of the set and iterates

```
(INFORM
:sender  (agent-identifier :name Treatment1@arwen:1099/JADE
  :addresses (sequence http://arwen:7778/acc )
  :X-JADE-agent-classname ofai.jade.examples.hospit al.
     KneeTreatmentAgent)
:receiver  (set ( agent-identifier :name Planner1@arwen:1099/JADE
  :addresses (sequence http://arwen:7778/acc )
  :X-JADE-agent-classname ofai.jade.examples.p lanner.PlannerAgent))
:reply-with  Planner1@arwen:1099/JADE1175078657404
:in-reply-to  R1175078657345_0  :language  fipa-sl
:ontology  DTCOntology  :protocol  DTCConfirmCommitment
:conversation-id  C14043096_1175078657345 )
```

**Listing 3.9:** A confirmation of the commitment specified in listing 3.8.

over each method of the schedule. For each item, it requests the commitment provider for this method to confirm its commitment at the start and finish times calculated by the scheduler. If the commitment provider refuses this request, the schedule containing the corresponding method is treated as not executable, and the whole procedure restarts with the next schedule candidate. If all commitment providers confirm their commitment, a feasible schedule has been found. Confirming a commitment means that the confirming agent is expected to execute the method within the right timeslot. A valid request for confirmation contains the predicate *DTCConfirmCommitment*. A valid answer is an empty `INFORM` message in the ontology *DTCOntology* in reply to the request, with matching conversation-IDs. Listings 3.8 and 3.9 show a confirmation request and the corresponding answer message.

### Communicate Expected Result Qualities

After a feasible schedule has been found, the planning, scheduling, and execution monitoring agent replies to the requester of the scheduled action execution with an `AGREE` message containing the predicate *DTCResult* providing information about expected start time, finish time, quality, cost, and duration. Listing 3.10 gives an example of such a reply to the `REQUEST` message shown in listing 3.3.

```
(AGREE
:sender  (agent-identifier :name Planner1@arwen:1099/JADE
  :addresses (sequence http://arwen:7778/acc )
  :X-JADE-agent-classname ofai.jade.examples.planner. PlannerAgent)
:receiver  (set (agent-identifier :name Requester@arwen:1099/JADE
  :addresses (sequence http://arwen:7778/acc )
  :X-JADE-agent-classname ofai.jade.examples. hospital.Requester ))
:content  "((DTCResult
  (Distribution :points (sequence 1.175078657E9) :probabilities (
     sequence 1.0))
  (Distribution :points (sequence 1.175078658E9) :probabilities (
     sequence 1.0))
  (Distribution :points (sequence 1.0) :probabilities (sequence 1.0))
  (Distribution :points (sequence 1.0) :probabilities (sequence 1.0))
  (Distribution :points (sequence 1.0) :probabilities (sequence 1.0))))
     "
:reply-with  Requester@arwen:1099/JADE1175078657449
:language  fipa-sl
:ontology  DTCOntology  :protocol  DTCSchedule
:conversation-id  ScheduleID267907171175078654618 )
```

**Listing 3.10:** Agree on the scheduling request in reply to the request in listing 3.3

```
(INFORM
:sender  (agent-identifier :name Treatment1@arwen:1099/JADE
  :addresses (sequence http://arwen:7778/acc )
  :X-JADE-agent-classname ofai.jade.examples.hospital.
     KneeTreatmentAgent )
:receiver  (set ( agent-identifier :name Planner1@arwen:1099/JADE
  :addresses (sequence http://arwen:7778/acc )
  :X-JADE-agent-classname ofai.jade.examples.p lanner.PlannerAgent ))
:content  "((DTCMethodExecutionResult C16586768_1175078657082
   1175078658))"
:language  fipa-sl  :ontology  DTCOntology  :protocol
   DTCMethodExecution
)
```

**Listing 3.11:** Indicate finished method execution

```
(INFORM
:sender  (agent-identifier :name Planner1@arwen:1099/JADE
  :addresses (sequence http://arwen:7778/acc )
  :X-JADE-agent-classname ofai.jade.examples.planner. PlannerAgent )
:receiver  (set ( agent-identifier :name Requester@arwen:1099/JADE
  :addresses (sequence http://arwen:7778/acc )
  :X-JADE-agent-classname ofai.jade.examples. hospital.Requester ) )
:content  "((DTCResult
  (Distribution :points (sequence 1.175078657E9) :probabilities (
     sequence 1.0))
  (Distribution :points (sequence 1.175078658E9) :probabilities (
     sequence 1.0))
  (Distribution :points (sequence 1.0) :probabilities (sequence 1.0))
  (Distribution :points (sequence 1.0) :probabilities (sequence 1.0))
  (Distribution :points (sequence 1.0) :probabilities (sequence 1.0))))
     "
:reply-with  Requester@arwen:1099/JADE1175078657449
:language  fipa-sl  :ontology  DTCOntology  :protocol  DTCSchedule
:conversation-id  ScheduleID267907171175078654618 )
```

**Listing 3.12:** An indication of finished schedule execution in reply to the request in listing 3.3.

### Execution Monitoring

During this phase the planning, scheduling, and execution monitoring agent waits for `INFORM` messages from schedule executing agents marked with protocol name *DTCMethodExecution* and specified in the ontology *DTCOntology* containing the predicate *DTCMethodExecutionResult*. If these messages arrive in the right order at the right times, it can proceed with informing the requester of the scheduled action execution that execution is proceeding as planned. Listing 3.11 shows an example of such an `INFORM` message.

### Result Notification

The planning, scheduling, and execution monitoring agent can send an `INFORM` message in reply to the scheduled action execution request to the scheduled action requester according to the FIPA Request Interaction Protocol, indicating that schedule execution

has finished successfully. This reply message must contain the predicate *DTCResult*. According to the actual finish time it may alter the expected result qualities and send them as final result qualities. Listing 3.12 shows an example result notification in reply to the request specified in listing 3.3.

### Exceptions to Protocol Flow

This scheduling protocol could be regarded as a three-layered hierarchical architecture, with the *scheduled execution requesting* layer on top, the *planning, scheduling, and execution monitoring* layer in the middle, and the *method execution* layer at the bottom. Successfull protocol flow across these layers is well defined via FIPA interaction protocols. It must also be possible to propagate failures through these layers. Errors in the top-down direction are propagated via the *FIPA Cancel Meta-Protocol*. `REQUEST` messages also flow in the top-down direction. `CANCEL` messages are expected to be identical to the corresponding `REQUEST` messages except for the performative that is changed from `REQUEST` to `CANCEL`. Therefore, cancellations are always non-ambiguous. `INFORM` messages are always sent in the bottom-up direction. If an error in the bottom-up direction has to be reported, the `INFORM` performative is simply replaced by a `FAILURE` performative. Consequently, `FAILURE` messages can also be always assigned unequivocally.

Failures at the method execution level occur when the executing agent is not able to meet the requested execution requirements. Such an error could be recovered by the planning, scheduling, and execution monitoring layer. For example, it could switch to another schedule or use slack times. If the error is not recoverable, a `FAILURE` message has to be sent to the *scheduled execution requesting* layer.

Cancellations may occur in *scheduled execution requesting* layer if the requester agent does not need the action to be performed any more, or at planning, scheduling, and execution monitoring level. There, `CANCEL` messages are used to dissolve commitments. Cancellations can be common during schedule selection, if an agent refuses to confirm its earlier commitment, and also during execution monitoring, if an agent fails to execute its TÆMS method (either altogether, or within expected performance bounds).

## 3.2. Integrating VIE-CDS with JADE

In this section, we outline our reference implementation for JADE. We do not go into great detail, because the *javadoc* generated documentation is better suited to provide implementation specifics. Here, we mainly want to point out the sometimes complex interrelationships between behaviours.

The appraoch taken is to describe scheduling functionality in the form of Java interfaces. On the one hand, this has the advantage that agents can use scheduling features, but still be part of another inheritance hierarchy. On the other hand, scheduling functionality does not depend on a particular TÆMS encoding. We provide an implementation of these interfaces for XTAEMS, because of its standard compliance and our familiarity with XML related tools.

### 3.2.1. Package Structure

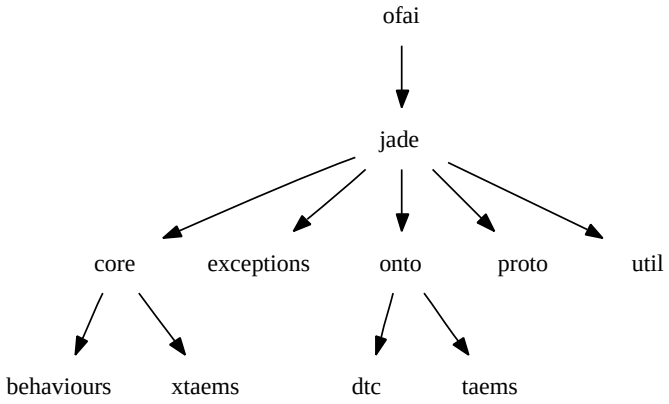


**Figure 3.9.:** Java package structure for our JADE reference implementation.

We make extensive use of Java's packaging mechanism to structure our code into semantically coherent groups, as illustrated in figure 3.9. Package `ofai.jade` is our top-level package and encapsulates all other sub-packages. It does not directly contain any Java classes. In the following, we shortly describe the package structure below the top-level package.

**core** contains interfaces that must be implemented. These interfaces define callback methods necessary to indicate scheduling specific events.

**core.behaviours** consists of abstract behaviours which must be subclassed by concrete implementations for a certain TÆMS encoding. Encoding independent classes are also contained but not declared as `abstract`. None of the behaviours in this package are communication related.

**core.xtaems** provides XTAEMS implementations for all interfaces defined in the package `core` and for all abstract classes defined in the package `core.behaviours`.

**exceptions** contains application specific exceptions.

**onto** provides ontology definitions.

**onto.dtc** contains the classes implementing *DTCOntology*.

**onto.taems** contains classes implementing *TAEMSOntology*.

**proto** consists of communication-specific behaviours.

**util** groups recurrent tasks not directly related to scheduler integration. It mostly contains conversion methods.

## 3.2.2. Implementation Concept

### Managing TÆMS structures

In the same manner that every standard JADE agent has an associated *content manager*, every agent which wants to participate in DTC-scheduled action execution must own a *TAEMSManager*. This manager is defined through the interface `TAEMSManager` allows the registering and unregistering of TÆMS structures at DF. Without requiring any further agent interaction, it publishes the agent's TÆMS knowledge with the Yellow Pages service. When an agent requests a commitment proposal, a commitment confirmation, or a method execution cancellation the manager communicates with the agent via callback functions defined in `TAEMSProviderCallbacks`.

Another important point is the way how TÆMS methods and JADE's action model are brought together. Here again `TÆMSManager` comes into play. After each local TÆMS structure modification, it asks the agent which behaviour a certain TÆMS method should be mapped to. If the agent does not provide a behaviour for a given TÆMS method, this method is treated as non-local and therefore not published with the Yellow Pages service.

### Request Scheduled Action Execution

A scheduling request is triggered by adding an `DTCScheduleInitiator` behaviour to the agent's behaviour queue. This behaviour subclasses JADE's *SimpleAchieveREInitiator* behaviour. When the behaviour is scheduled by the agent, it automatically queries the directory facilitator for a DTC scheduling service. It then randomly chooses one agent providing this service and sends a scheduling request message, as described in section 3.1.3. Afterwards, it waits for responses and notifies the agent about the current state of handling of the scheduling request via the callback functions defined in `DTCRequesterCallbacks`.

### Planning, Scheduling, Execution Monitoring

This task corresponds to the responsiblities of the planning, scheduling, execution monitoring role illustrated in figure 3.8. It is implemented as nested finite state machine behaviours triggered by the listener behaviour `DTCScheduleRequestResponder` which provides an implementation of JADE's *AchieveREResponder* behaviour. This responder behaviour again uses two behaviours: one for handling scheduling requests, and one for providing the result messages. The result message providing behaviour is discussed later on, because it corresponds to the *Execution Monitoring* protocol phase.

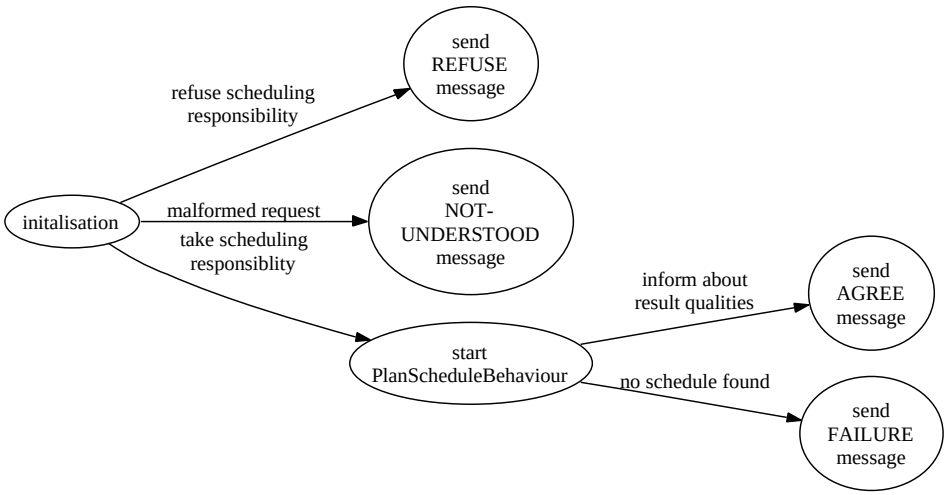


**Figure 3.10.:** States and transitions of the `HandleRequestBehaviour` finite state machine.

Our scheduling request handling behaviour is implemented by `HandleRequestBehaviour`. This behaviour is a finite state machine behaviour. Its states and transitions are illustrated in figure 3.10. The *initialisation* state carries out some JADE-specific initialisations and asks the agent whether a valid scheduling and execution monitoring request should be accepted. If so, then a `PlanScheduleBehaviour` is added to the agent's behaviour queue. This behaviour is again a finite state machine behaviour with the states and transitions illustrated in figure 3.11. If this behaviour can find a feasible schedule, an `AGREE` message to the scheduled action execution requester is sent. Otherwise a `FAILURE` message is sent.

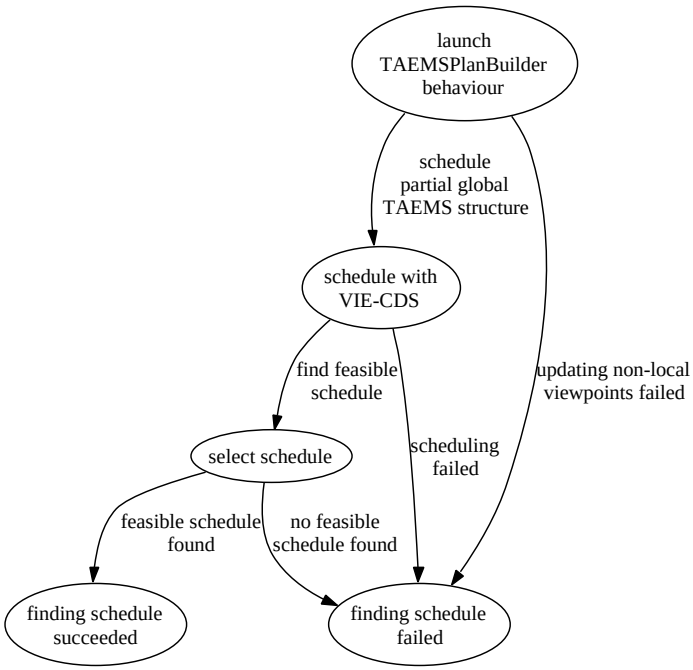


**Figure 3.11.:** `PlanScheduleBehaviour`'s states and transitions

`PlanScheduleBehaviour` firstly starts the behaviour `TAEMSPlanBuilder` which deals with the task of updating non-local viewpoints. After a partial global TÆMS structure has been built, this structure is fed to VIE-CDS. VIE-CDS returns a set of possible schedules. Running VIE-CDS is part of the *Scheduling* protocol phase. Out of this set the state *select schedule* tries to select a feasible schedule according to the protocol phase *Schedule Selection* described in section 3.1.3. After that, an `AGREE` message is sent to the requester.

Behaviour `TAEMSPlanBuilder` deals with protocol phase *Updating Non-Local Viewpoints* and partially with phase *Scheduling* (see section 3.1.3). `TAEMSPlanBuilder` again is a

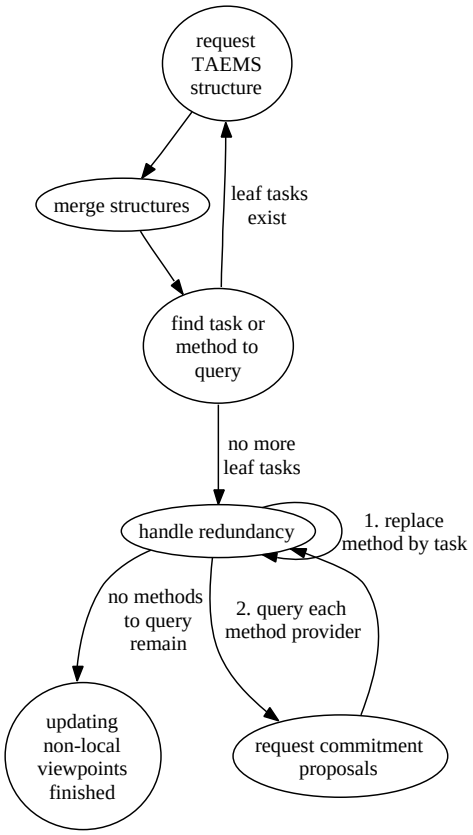


**Figure 3.12.:** States and transitions of the `PlanScheduleBehavior` finite state machine.

finite state machine behaviour with its states and transistions illustrated in figure 3.12. State *request TAEMS structure* implements *AchieveREInitiator* to query other agents for partial TÆMS structures. At the very first call it requests structures describing the scheduling goal. Afterwards all retrieved partial TÆMS structures get merged following the guidelines of *Updating Non-Local Viewpoints* in section 3.1.3. The next state searches TÆMS tasks having no child tasks or methods in the already built TÆMS structure. If such tasks or methods are found, the finite state machine returns to the first state and requests partial TÆMS structures containing these tasks or methods. If no tasks or methods to be requested remain, this behaviour deals with redundant TÆMS method providers. Each method $m$ in the already built partial global TÆMS structure is replaced by a task $t$ having the same ID as method $m$. Then it queries each method provider for a commitment proposal for method $m$. Each returned proposal for method $m$ is added as a child method $m_i$ of task $t$, where $0 \leq i < n$, $i \in N_0$ and $n$ is the number of returned proposals. We use the quality accumulation function `exactly_one` expressing that only one method provider can be selected. To ensure uniqueness of method IDs, the created methods $m_i$ are labelled after the schema

`<globally unique method provider ID>#<method ID>`

where the globally unique method provider ID is the string representation of JADE's agent ID and method ID is the method's original ID.

The agent playing the TAEMS Provider role must add the behaviours `TAEMSSendResponder`, `TAEMSKnowledgeAboutResponder`, and `DTCConfirmCommitmentResponder` to its behaviour queue for handling protocol phases *Updating Non-Local Viewpoints*, *Scheduling*, and *Schedule Selection*. Each one of these three behaviours implements JADE's *AchieveREResponder* behaviour. `TAEMSSendResponder` directly communicates with `TAEMSManager` to retrieve necessary TÆMS structures without involving the agent. `TAEMSKnowledgeAboutResponder` and `DTCConfirmCommitmentResponder` communicate with the agent through callback functions defined in the interface `TAEMSProviderCallbacks`.

### Scheduled Action Execution

The phases *Execution Monitoring* and *Result Notification* are implemented in `HandleResultBehaviour` for the agent role planning, scheduling, and execution monitor-

ing. This behaviour is a handler behaviour for JADE's *AchieveREResponder* behaviour. It is a simple finite state machine behaviour, illustrated in figure 3.13. This behaviour periodically searches the agent's incoming message queue for messages pertaining to the *Execution Monitoring* phase. If all notifications about TÆMS method executions arrive in the right order and their finish times fall within the temporal boundaries set by VIE-CDS, this finite state machine behaviour switches to state *send INFORM message* and sends an `INFORM` message according to the *Result Notifaction* phase. Otherwise, a `FAILURE` message is sent.

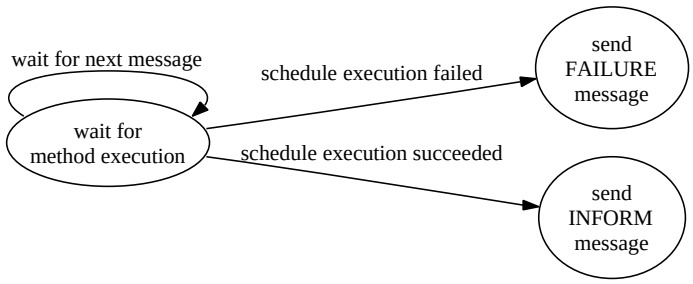


**Figure 3.13.:** States and transitions of the `HandleResultBehaviour` finite state machine.

Execution of TAEMS methods occurs in the agent role TAEMS Provider. When the behaviour `DTCConfirmCommitmentResponder` receives a request for commitment confirmation, it adds the behaviour `DTCMethodExecutionLauncher` to the agent's behaviour queue. The first state is a *WakerBehaviour* which blocks the launcher behaviour until the commited TÆMS method's execution start time. Then the agent is asked via a callback function defined in the interface `DTCExecutionCallbacks` if the start should be delayed. The *check delay* state checks whether the delayed start lies past the last possible start time. The last possible start time is the greatest point in the start time distribution. If the delay is acceptable the behaviour returns to the *delay* state. If the last possible start time has passed a `FAILURE` message according to the *Execution Monitoring* phase is sent. If no further delay is requested and no error has occured, the behaviour registered at *TAEMSManager* for a the specific TÆMS method is scheduled. If the behaviour terminates successfully, the executing agent sends the expected `INFORM` message to the planning, scheduling, and execution monitoring agent. Otherwise, it sends a `FAILURE` message.

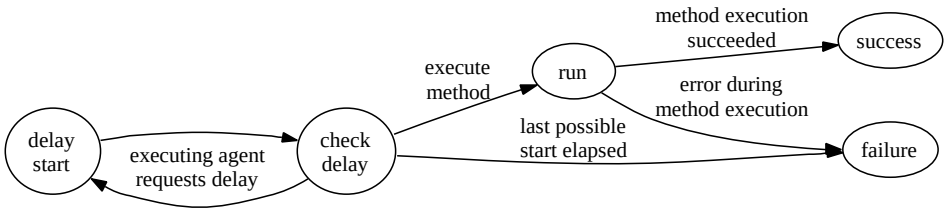


**Figure 3.14.:** States and transitions of the `DTCMethodExecutionLauncher` finite state machine.

### 3.2.3. XTAEMS Agent

The class `XTAEMSAgent` in package `ofai.jade.core.xtaems` provides an agent implementation with all functionality necessary to participate in DTC scheduled action execution. Listing 3.13 shows an excerpt of this agent class. This is the minimum setup work necessary for an agent to be able to deal with TÆMS structures. If we want the agent to be able to carry out the scheduling task, we only need to add the code snippet shown in listing 3.14 to the agent's setup method.

One way of using our reference implementation is to inherit from class `XTAEMSAgent`. If the agent is already involved in another inheritance hierarchy, it is still possible to use DTC scheduled action execution with XTAEMS, by simply adding all necessary behaviours to the agent's setup method, as shown in listings 3.13 and 3.14 to the agent's setup method. The behaviour constructors require some callback objects to be passed. The easiest way is to implement them directly in the agent class, but in principle it is up to the agent designer where to implement these callbacks.

```
@Override
 protected void setup() {
   super.setup();
   logger = jade.util.Logger.getMyLogger(getAID().toString());
   TAEMSManager manager = new XTAEMSManager(logger);
   manager.setTAEMSCallbackHandler(this);
   setTAEMSManager(manager);
   getContentManager().registerOntology(ontologyTAEMS);
   getContentManager().registerOntology(ontologyDTC);
   DFService.register(this, getDFAgentDescription());
   registerTAEMS();
   addBehaviour(new TAEMSSendResponder(this));
   addBehaviour(new TAEMSKnowledgeAboutResponder(this));
   addBehaviour(new DTCConfirmCommitmentResponder(this, this, this));
   addBehaviour(new TAEMSCancelListener(this, this));
 }
```

**Listing 3.13:** A minimum setup for an XTAEMS Agent without exception handling.

```
// Add DTC Scheduling capabilites
try {
  addBehaviour(new DTCScheduleRequestResponder(this, this));
} catch (TAEMSAgentRequiredException e) {
  e.printStackTrace();
  doDelete();
}
```

**Listing 3.14:** Addition of scheduling functionality

# 4. Example Usage

Accompanying to the reference implementation, a small example application to demonstrate the elementary functionality of DTC scheduling integration has been developed. Beforehand, we want to emphasise that this example only serves demonstration purposes and does not claim any medical correctness or completeness, nor any other immediate suitability for real-word application. The main intention is to demonstrate how even during scheduled action execution an agent keeps full control over its behaviours, despite the complex communication flow in the background.

## 4.1. Implementation

All agents in this example are direct subclasses of the `XTAEMS` agent discussed in section 3.2.3. Figure 4.1 illustrates where the example code is located in our package structure. Agents offering a graphical user interface make use of behaviour `SWTGUIBehaviour`, which gives access to the SWT[1] graphics toolkit. In this way, we consistently apply JADE's principle to implement everything an agent does as a behaviour.

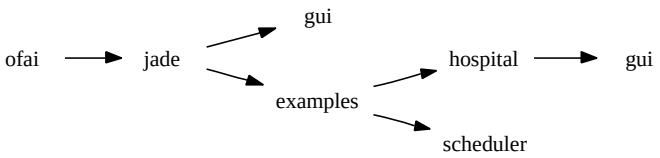


**Figure 4.1.:** Java package structure for our example.

The following three agent types are implemented:

[1] http://www.eclipse.org/swt/, last visited May 2, 2007.

**TreatmentAgent** owns TÆMS structures describing medical procedures and provides no graphical user interface. It resides in package `ofai.jade.examples.hospital`.

**RequesterAgent** owns TÆMS structures, knows how to query DF for available TÆMS tasks, and implements the initiator role of the `DTCSchedule` protocol. It also provides a graphical user interface for monitoring schedule execution status. RequesterAgent is also implemented in package `ofai.jade.examples.hospital`. Figure 4.2 shows the graphical user interface of `RequesterAgent`. It maintains a list of requested scheduled action executions since start-up. A requested schedule may have one of four possible status codes, where 1 means a schedule has been requested, 2 indicates that a feasible schedule has been found, and 3 marks successfully executed schedules. Status code 4 reports an error. `RequesterAgent` allows to browse TÆMS domains and the tasks registered with the DF, as illustrated in figure 4.3. The criteria sliders shown in figure 4.4 allow the specification of scheduling criteria for a certain scheduling request.

**SchedulerAgent** does not own any TÆMS structures but provides the planning, scheduling, and execution monitoring service. It resides in package `ofai.jade.examples.scheduler`. Figure 4.5 pictures the `SchedulerAgent`'s graphical user interface. It allows browsing all requested scheduled action executions. For each scheduling request, it provides the corresponding partial global XTAEMS and TTAEMS structures, their graphical representation, all possible schedules for this structure, and the schedule selected.

Since we did not specify where to take TÆMS knowledge from, we simply read local TÆMS structures at agent start-up from a file. It would also be possible for a domain specific problem-solver to generate local TÆMS structures which would be registered with the agent's TAEMSManager.

TÆMS and our integration approach do not explicitely specify how time should be encoded. In our sample scenario we use UNIX timestamps. Therefore, our schedule resolution is one second. The class `JadeSchedulerHelper` in package `ofai.jade.util` provides methods to convert between Java's internal time representation in milliseconds, Java *Date* objects, and UNIX timestamps.

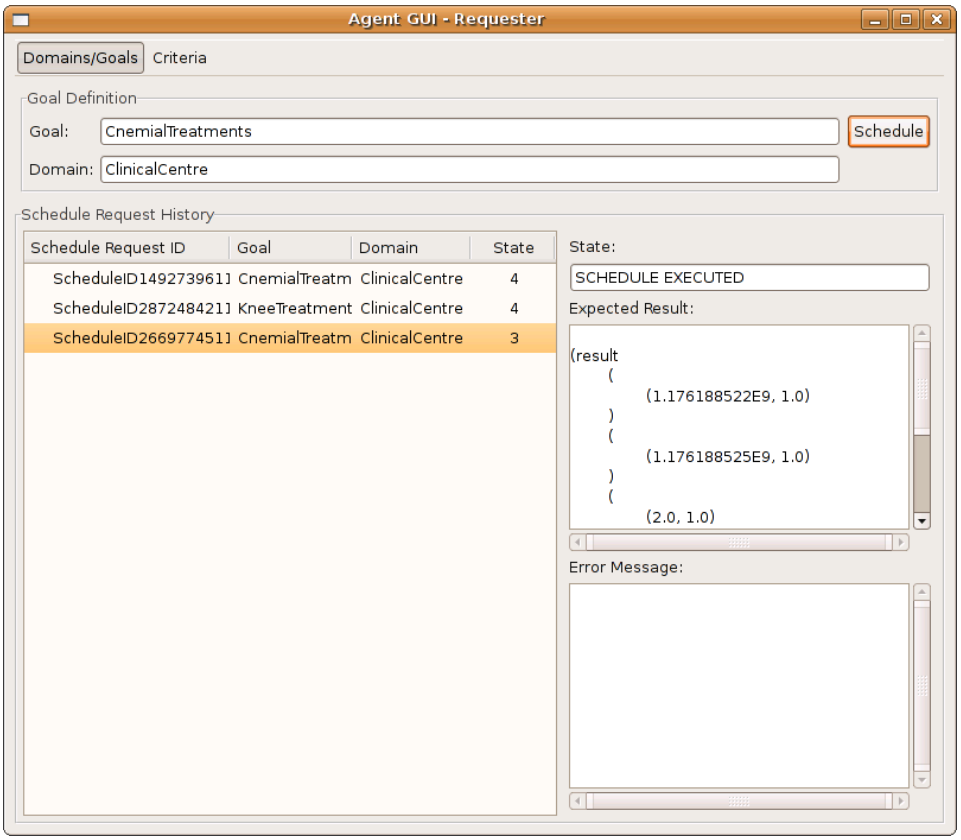


**Figure 4.2.:** The `RequesterAgent`'s graphical user interface.


**Figure 4.3.:** The `RequesterAgent`'s possible scheduling goals browser.

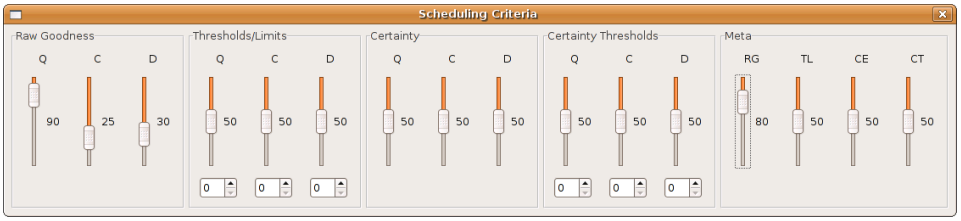


**Figure 4.4.:** The`RequesterAgent`'s criteria sliders.

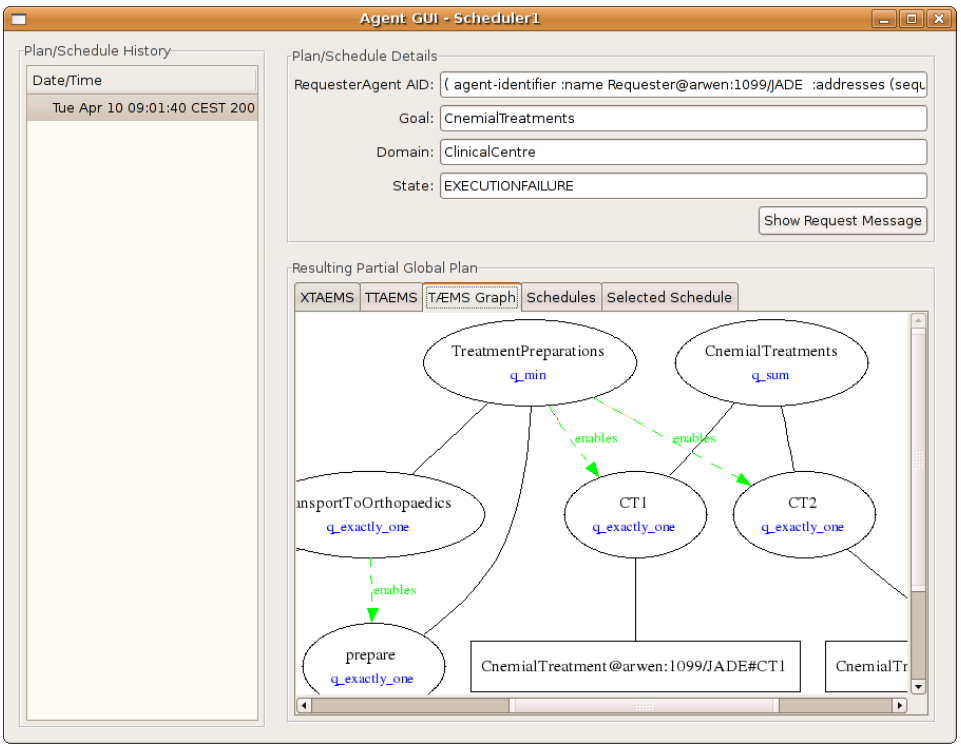


**Figure 4.5.:** The `SchedulerAgent`'s graphical user interface.

## 4.2. Scenario

We assume our application to be part of a hospital-wide multi-agent system using JADE as development framework. Each medical care unit has a networked computer system assigned running a JADE agent platform. Inside this platform several agents with TÆMS knowledge, several agents with DTC scheduling capabilities and a DF exist. TÆMS knowledge agents are interconnected with surgeries and administrative staff.

TÆMS agents have knowledge about medical procedures encoded in TÆMS structures. They provide no graphical interface because they gain necessary information from other hospital information resources. The desk personnel is provided with a graphical user interface for ordering medical treatments and monitoring their progress. TÆMS agents with scheduling capabilities also have a graphical user interface, which is accessible for administrative purposes only.

In our scenario, we instantiate five `TreatementAgents` called *Preparator*, *AnkleTreatment*, *CnemialTreatment*, *KneeTreatment1* and *KneeTreatment2*, two `SchedulerAgents`

```
<?xml version="1.0" encoding="UTF-8"?>
  <taems>
    <task id="TreatmentPreparations" qaf="min">
      <subtask ref="transportToOrthopaedics" />
      <subtask ref="prepare" />
    </task>
    <enables from="transportToOrthopaedics" to="prepare"
      id="bringBeforePrepare" />
    <method id="transportToOrthopaedics" />
    <method id="prepare" />
  </taems>
```

**Listing 4.1:** The *Preparator*'s local TÆMS structure.

called *Scheduler1* and *Scheduler2*, and one `RequesterAgent` called *Requester*. *Preparator* has knowledge about how to transport patients to the orthopaedics care unit and how to prepare them for treatment. Listing 4.1 shows its local TÆMS structure. Listings 4.2, 4.3, 4.4 and 4.5 list the local TÆMS structures of the *AnkleTreatment*, *CnemialTreatment*, *KneeTreatment1*, *KneeTreatment2* and *Requester* agents.

### 4.2.1. Sample Run

We want the TÆMS task `KneeTreatments` to be scheduled twice. The first time, we set all criteria sliders to value 50. Then we press the button *Schedule*. *Scheduler2* takes over scheduling responsibilities. It builds the partial global TÆMS structure shown in figure 4.6. The schedule shown in listing 4.6 with quality 2, cost 3 and duration 3 is selected. The second time, we set quality to 90, cost to 30, and duration to 30 in the raw goodness of scheduling criteria, because we want the quality of scheduled action execution to be more important than cost or duration. This time *Scheduler1* does the scheduling and execution monitoring job. It builds the same partial global TÆMS structure as in the first run, but this time the selected schedule shown in listing 4.7 is different. The schedule has quality 3, cost 4, and duration 4. As expected, this schedule trades in longer duration and higher costs for improved quality.

```
<?xml version="1.0" encoding="UTF-8"?>
<taems>
  <task id="AnkleTreatments" qaf="sum">
    <subtask ref="AT1" />
    <subtask ref="AT2" />
  </task>
  <method id="AT1" />
  <method id="AT2" />
  <task id="TreatmentPreparations" qaf="min" />
  <enables from="TreatmentPreparations" to="AT1"
    id="enableAT1" />
  <enables from="TreatmentPreparations" to="AT2"
    id="enableAT2" />
</taems>
```

**Listing 4.2:** The *AnkleTreatment*'s local TÆMS structure.

```
<?xml version="1.0" encoding="UTF-8"?>
<taems>
  <task id="CnemialTreatments" qaf="sum">
    <subtask ref="CT1" />
    <subtask ref="CT2" />
  </task>
  <method id="CT1" />
  <method id="CT2" />
  <task id="TreatmentPreparations" qaf="min" />
  <enables from="TreatmentPreparations" to="CT1"
    id="enableCT1" />
  <enables from="TreatmentPreparations" to="CT2"
    id="enableCT2" />
</taems>
```

**Listing 4.3:** The *CnemialTreatment*'s local TÆMS structure.

```
<?xml version="1.0" encoding="UTF-8"?>
<taems>
  <task id="KneeTreatments" qaf="sum">
    <subtask ref="KT1" />
    <subtask ref="KT2" />
  </task>
  <method id="KT1" />
  <method id="KT2" />
  <task id="TreatmentPreparations" qaf="min"/>
  <enables from="TreatmentPreparations" to="KT1"
    id="enableKT1" />
  <enables from="TreatmentPreparations" to="KT2"
    id="enableKT2" />
</taems>
```

**Listing 4.4:** The *KneeTreatment1*'s and *KneeTreatment2*'s local TÆMS structure.

```
<?xml version="1.0" encoding="UTF-8"?>
<taems>
  <task id="FullCheck" qaf="sum_all">
    <subtask ref="KneeTreatments" />
    <subtask ref="AnkleTreatments" />
    <subtask ref="CnemialTreatments" />
  </task>
</taems>
```

**Listing 4.5:** The *Requester*'s local TÆMS structure.

```
1. Preparator@arwen:1099/JADE#transportToOrthopaedics at 1176192116
2. Preparator@arwen:1099/JADE#prepare at 1176192117
3. KneeTreatment2@arwen:1099/JADE#KT1 at 1176192118
```

**Listing 4.6:** The first schedule with quality 2, cost 3 and duration 4.

```
1. Preparator@arwen:1099/JADE#transportToOrthopaedics at 1176192188
2. Preparator@arwen:1099/JADE#prepare at 1176192189
3. KneeTreatment2@arwen:1099/JADE#KT2 at 1176192190
4. KneeTreatment2@arwen:1099/JADE#KT1 at 1176192191
```

**Listing 4.7:** The second schedule with quality 3, cost 4 and duration 4.

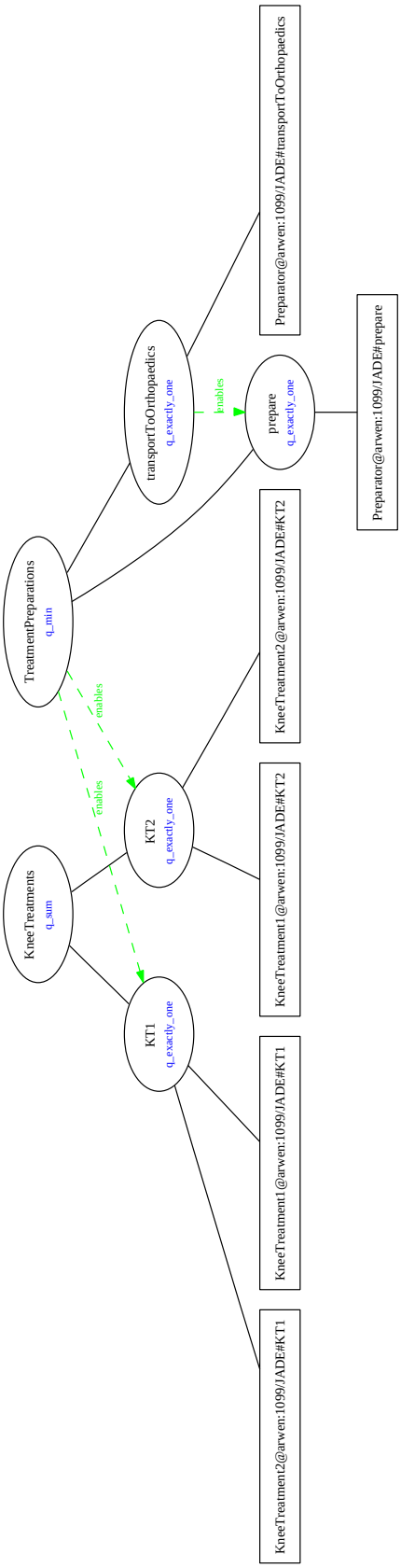


**Figure 4.6.:** Partial global TÆMS structure for `KneeTreatments`

# 5. Conclusions and Further Work

This chapter overviews related work with similar goals, but different approaches. It summarises what we have done so far and additionally outlines what could or should be done in the future to improve the integration of criteria-driven scheduling with FIPA compliant multi-agent platforms.

## 5.1. Related Work

### 5.1.1. Distributed Sliding Window Scheduler

Scott Logie et al. have developed a sliding window scheduling method [Logie et al., 2003] based on the recursive propagation technique presented in [Hino et al., 2001]. The main idea of the recursive propagation technique is to send schedule changes from one agent owning tasks included in the schedule to another. When no agents to be notified remains, the cumulative result of changes is sent back to the change message initiator. In contrast to [Hino et al., 2001], to save save computation and communication costs notifications of task re-ordering are only propagated inside a time window $[t_1, t_2)$, where $t_1$ denotes the window's start time and $t_2$ the window's end time.

In [Logie et al., 2004] an integration approach of sliding window scheduling with JADE is proposed. A *scheduling agent* (JSA) is a multi-threaded Java application outside the JADE agent platform. This scheduling agent communicates through sockets with a JADE agent. Together, these two agents together form a so called *composite scheduling entity* (CSE). The JSA is again divided into sub-agents carrying out the scheduling job or exchanging messages with the assigned JADE agent.

This model provides a fully distributed multi-agent scheduling approach. The real scheduling work is done outside JADE, which allows moving computation intensive

scheduling tasks to powerful computers. It introduces a new communication dependency not covered by any platform specification.

### 5.1.2. Parma Development Environment (PARADE)

The *PARADE* (Parma Development Environment) toolkit

> *" is providing the agent developer with a hybrid agent architecture capable of promoting inter-operability and supporting autonomy exploiting the semantics of FIPA ACL. Such an architecture is basically goal-oriented but it also integrates reactive behaviours."*
>
> [Bergenti and Poggi, 2001, p.633]

*PARADE* provides semantic inter-operability based on already defined FIPA interaction protocols which originally only allow syntactic inter-operability. The main idea is to describe agents in terms of mental states, a set of possible actions, and a set of FIPA interaction protocols. Beliefs, persistent goals, and transient goals are all described by propositions composing the mental state of an agent.

The planning engine must be provided with all actions an agent playing a certain role can be requested to execute and all interaction protocols supported by this agent. Each action and protocol has *pre-conditions* and *post-conditions* expressed as propositions assigned. When an agent wants to achieve a certain persistent goal the planning engine tries to find a sequence of actions and interaction protocols sufficing pre-conditions and post-conditions of used actions and protocols. Standard planning techniques can be used for that task.

In principle, *PARADE* relies on FIPA-only mechanisms and could be used on all FIPA compliant multi-agent platforms. At present only a version for JADE is implemented.

## 5.2. Conclusion

GPGP provides an elaborate framework which inspired the integration of criteria-driven scheduling in FIPA compliant multi-agent infrastructures. Since GPGP is tightly related to JAF, it was not possible to transfer all ideas from it to our integration concept.

JADE is a well-known FIPA-compliant multi-agent development framework. The definition of a XML-based encoding of TÆMS structures provides the basis for our integration work. It alters TÆMS from the main data and communication structure used in JAF to an abstract description language for problem decomposition in multi-agent planning and scheduling scenarios. VIE-CDS has been extended to understand this encoding. Furthermore, VIE-CDS gained a new API making it easier to support multiple TÆMS encodings as well as an API allowing the implementation of multiple output formats.

The whole integration process is driven by the idea to use FIPA defined platform features where possible and to reduce TÆMS' function to problem structure definition. This leads to the definition of scheduled action execution as a pre-determined sequence of ACL-messages based on FIPA standardised interaction protocols. TÆMS knowledge is published at the Directory Facilitator. Ontologies for TÆMS and DTC scheduling are developed to provide the vocabulary necessary for ACL message exchange. Scheduling protocol flow is structured in five phases:

1. **Scheduling request**: An agent selects a scheduling service and requests scheduled action execution.
2. **Updating non-local viewpoints**: During this phase, a partial global TÆMS structure describing the problem structure is built.
3. **Scheduling**: A design-to-criteria scheduler processes the partial global TÆMS structure and tries to find a set of possible schedules. In our case, this is VIE-CDS.
4. **Schedule Selection**: In this phase, the scheduling service tries to find a feasible schedule out of the possible schedules.
5. **Execution monitoring:** The scheduling service waits for notification of finished TÆMS method executions.

The scheduling request (first phase) is based on the *FIPA Request Interaction Protocol*. This protocol wraps all other phases. Since agreeing on a request and result notification are part of this interaction protocol, we directly use these protocol states to communicate expected and final schedule execution qualities. Failures in protocol flow are propagated back to the scheduled action execution requester agent. This back-propagation

is already defined by FIPA through the exceptions to protocol flow in their interaction protocols.

The probably largest differences to the scheduling approaches presented in section 5.1 are the explicit representation of *qualitative* (coordination dependencies) and *quantitative* (probability distributions) aspects of coordination, as well as, and this is maybe the most important one, criteria-driven schedule selection. The scheduler is able to make trade-offs between quality, cost, and duration of schedule execution according to user preferences.

In software development terms, this integration approach would be classified as pre-alpha. It should be treated as a first step towards really usable criteria-driven scheduling in an FIPA compliant multi-agent infrastructure.

## 5.3. Further Work

One big disadvantage of our integration approach at conceptual level is the embedding of TÆMS structures in ACL-messages as non-accessible blocks for standard FIPA content language manipulation mechanisms. Consequently, a next required step would be to define an ontology which allows the definition of TÆMS structures using standard content language mechanisms. Generally speaking, it would be nice to better exploit the semantics of FIPA SL. The maybe tightest integration of VIE-CDS with JADE possible would be making VIE-CDS run on top of JADE's objects implementing the TÆMS ontology. Instead of providing a set of possible schedules it could provide an already instantiated finite state machine behaviour realising the schedule selection and execution monitoring jobs.

Another disadvantage at implementation level is given by the fact that an agent cannot deal with multiple scheduling requests concurrently. The behaviour providing the planning, scheduling, and execution monitoring services must wait for the currently processed schedule to be finished to be able to handle a new request. This could be solved by altering the implementation to support multiple scheduling request sessions. Section 5.4.1 in [Bellifemine et al., 2007] indicates a possible solution path for this problem.

Concerning the integration with JADE, moving criteria-driven scheduling support from agent level to kernel-service level should also be considered. This could improve performance, but would probably break interoperability with other FIPA-compliant frameworks.

# A. XTAEMS Examples

Here we present some XTAEMS examples in direct comparison to their corresponding TTAEMS encodings.

## A.1. The Getting Dinner Example

This is the *Getting Dinner* example from the TÆMS white paper [Horling et al., 1999]. Agent *Me* is not declared in XTAEMS. For a detailed explanation see section 2.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<taems>
  <!--
    (spec_agent
    (label Me)
    )
    (spec_task_group
    (label Get-Dinner)
    (agent Me)
    (subtasks Get-Delivery Eat-Out Make-Dinner)
    (qaf q_exactly_one)
    )
  -->
  <task id="Get-Dinner" qaf="exactly_one">
    <subtask ref="Get-Delivery" />
    <subtask ref="Eat-Out" />
    <subtask ref="Make-Dinner" />
  </task>
  <!--
    (spec_task
    (label Get-Delivery)
    (agent Me)
    (subtasks Call-Restaurant)
```

```
  (supertasks Get-Dinner)
  (qaf q_exactly_one)
  )
-->
<task id="Get-Delivery" qaf="exactly_one">
  <subtask ref="Call-Restaurant" />
</task>
<!--
  (spec_task
  (label Eat-Out)
  (agent Me)
  (subtasks Go-To-Restaurant Order-Dinner)
  (supertasks Get-Dinner)
  (qaf q_seq_sum)
  )
-->
<task id="Eat-Out" qaf="seq_sum">
  <subtask ref="Go-To-Restaurant" />
  <subtask ref="Order-Dinner" />
</task>
<!--
  (spec_task
  (label Make-Dinner)
  (agent Me)
  (subtasks Obtain-Ingredients Prepare-Food Cook-Food)
  (supertasks Get-Dinner)
  (qaf q_seq_sum)
  )
-->
<task id="Make-Dinner" qaf="seq_sum">
  <subtask ref="Obtain-Ingredients" />
  <subtask ref="Prepare-Food" />
  <subtask ref="Cook-Food" />
</task>
<!--
  (spec_method
  (label Call-Restaurant)
  (agent Me)
  (supertasks Get-Delivery)
  (outcomes
  (Outcome_1
  (density 1.0)
```

```
  (quality_distribution 5.0 1.0)
  (duration_distribution 30 0.7 45 0.3)
  (cost_distribution 10 1.0)
  )
  )
  )
-->
<method id="Call-Restaurant">
  <outcome probability="1">
    <quality>
      <distribution point="5" probability="1" />
    </quality>
    <duration>
      <distribution point="30" probability="0.7" />
      <distribution point="45" probability="0.3" />
    </duration>
    <cost>
      <distribution point="10" probability="1" />
    </cost>
  </outcome>
</method>
<!--
  (spec_method
  (label Go-To-Restaurant)
  (agent Me)
  (supertasks Eat-Out)
  (outcomes
  (Outcome_1
  (density 1.0)
  (quality_distribution 1.0 1.0)
  (duration_distribution 10.0 1.0)
  (cost_distribution 0 1.0)
  )
  )
  )
-->
<method id="Go-To-Restaurant">
  <outcome probability="1">
    <quality>
      <distribution point="1" probability="1" />
    </quality>
    <duration>
```

```
      <distribution point="10" probability="1" />
    </duration>
    <cost>
      <distribution point="0" probability="1" />
    </cost>
  </outcome>
</method>
<!--
  (spec_method
  (label Order-Dinner)
  (agent Me)
  (supertasks Eat-Out)
  (outcomes
  (Outcome_1
  (density 1.0)
  (quality_distribution 10 0.6 8 0.4)
  (duration_distribution 60 0.8 40 0.2)
  (cost_distribution 20.0 1.0)
  )
  )
  )
-->
<method id="Order-Dinner">
  <outcome probability="1">
    <quality>
      <distribution point="10" probability="0.6" />
      <distribution point="8" probability="0.4" />
    </quality>
    <duration>
      <distribution point="60" probability="0.8" />
      <distribution point="40" probability="0.2" />
    </duration>
    <cost>
      <distribution point="20" probability="1" />
    </cost>
  </outcome>
</method>
<!--
  (spec_method
  (label Obtain-Ingredients)
  (agent Me)
  (supertasks Make-Dinner)
```

```
  (outcomes
  (Outcome_1
  (density 1.0)
  (quality_distribution 1.0 1.0)
  (duration_distribution 10.0 1.0)
  (cost_distribution 6.0 1.0)
  )
  )
  )
-->
<method id="Obtain-Ingredients">
  <outcome probability="1">
    <quality>
      <distribution point="1" probability="1" />
    </quality>
    <duration>
      <distribution point="10" probability="1" />
    </duration>
    <cost>
      <distribution point="6" probability="1" />
    </cost>
  </outcome>
</method>
<!--
  (spec_method
  (label Prepare-Food)
  (agent Me)
  (supertasks Make-Dinner)
  (outcomes
  (Outcome_1
  (density 1.0)
  (quality_distribution 3.0 1.0)
  (duration_distribution 15.0 1.0)
  (cost_distribution 0.0 1.0)
  )
  )
  )
-->
<method id="Prepare-Food">
  <outcome probability="1">
    <quality>
      <distribution point="3" probability="1" />
```

```
      </quality>
      <duration>
        <distribution point="15" probability="1" />
      </duration>
      <cost>
        <distribution point="0" probability="1" />
      </cost>
    </outcome>
  </method>
  <!--
    (spec_method
    (label Cook-Food)
    (agent Me)
    (supertasks Make-Dinner)
    (outcomes
    (Outcome_1
    (density 1.0)
    (quality_distribution 3.0 1.0)
    (duration_distribution 30.0 1.0)
    (cost_distribution 0.0 1.0)
    )
    )
    )
  -->
  <method id="Cook-Food">
    <outcome probability="1">
      <quality>
        <distribution point="3" probability="1" />
      </quality>
      <duration>
        <distribution point="30" probability="1" />
      </duration>
      <cost>
        <distribution point="0" probability="1" />
      </cost>
    </outcome>
  </method>
</taems>
```

## A.2. Schedules for the Getting Dinner Example

These are possible schedules generated by VIE-CDS for the *Getting Dinner* example in XTAEMS encoding.

```
<?xml version="1.0" encoding="UTF-8"?>
<schedules>
  <!--
    (spec_schedule
    (schedule_elements
    (Order-Dinner
    (start_time_distribution 0.0 1.0)
    (finish_time_distribution 40.0 0.2 60.0 0.8)
    (quality_distribution 8.0 0.4 10.0 0.6)
    (cost_distribution 20.0 1.0)
    (duration_distribution 40.0 0.2 60.0 0.8)
    )
    (Go-To-Restaurant
    (start_time_distribution 40.0 0.2 60.0 0.8)
    (finish_time_distribution 50.0 0.2 70.0 0.8)
    (quality_distribution 1.0 1.0)
    (cost_distribution 0.0 1.0)
    (duration_distribution 10.0 1.0)
    )
    )
    (start_time_distribution 0.0 1.0)
    (finish_time_distribution 50.0 0.2 70.0 0.8)
    (quality_distribution 9.0 0.4 11.0 0.6)
    (cost_distribution 20.0 1.0)
    (duration_distribution 50.0 0.2 70.0 0.8)
    (rating 0.6666666666666666)
    )
  -->
  <schedule rating="0.6666666666666666">
    <start>
      <distribution point="0.0" probability="1.0" />
    </start>
    <finish>
      <distribution point="50.0" probability="0.2" />
      <distribution point="70.0" probability="0.8" />
    </finish>
    <quality>
```

```
  <distribution point="9.0" probability="0.4" />
  <distribution point="11.0" probability="0.6" />
</quality>
<cost>
  <distribution point="20.0" probability="1.0" />
</cost>
<duration>
  <distribution point="50.0" probability="0.2" />
  <distribution point="70.0" probability="0.8" />
</duration>
<element id="Order-Dinner">
  <quality>
    <distribution point="8.0" probability="0.4" />
    <distribution point="10.0" probability="0.6" />
  </quality>
  <cost>
    <distribution point="20.0" probability="1.0" />
  </cost>
  <duration>
    <distribution point="40.0" probability="0.2" />
    <distribution point="60.0" probability="0.8" />
  </duration>
  <start>
    <distribution point="0.0" probability="1.0" />
  </start>
  <finish>
    <distribution point="40.0" probability="0.2" />
    <distribution point="60.0" probability="0.8" />
  </finish>
</element>
<element id="Go-To-Restaurant">
  <quality>
    <distribution point="1.0" probability="1.0" />
  </quality>
  <cost>
    <distribution point="0.0" probability="1.0" />
  </cost>
  <duration>
    <distribution point="10.0" probability="1.0" />
  </duration>
  <start>
    <distribution point="40.0" probability="0.2" />
```

```
      <distribution point="60.0" probability="0.8" />
    </start>
    <finish>
      <distribution point="50.0" probability="0.2" />
      <distribution point="70.0" probability="0.8" />
    </finish>
  </element>
</schedule>
<!--
  (spec_schedule
  (schedule_elements
  (Obtain-Ingredients
  (start_time_distribution 0.0 1.0)
  (finish_time_distribution 10.0 1.0)
  (quality_distribution 1.0 1.0)
  (cost_distribution 6.0 1.0)
  (duration_distribution 10.0 1.0)
  )
  (Cook-Food
  (start_time_distribution 10.0 1.0)
  (finish_time_distribution 40.0 1.0)
  (quality_distribution 3.0 1.0)
  (cost_distribution 0.0 1.0)
  (duration_distribution 30.0 1.0)
  )
  (Prepare-Food
  (start_time_distribution 40.0 1.0)
  (finish_time_distribution 55.0 1.0)
  (quality_distribution 3.0 1.0)
  (cost_distribution 0.0 1.0)
  (duration_distribution 15.0 1.0)
  )
  )
  (start_time_distribution 0.0 1.0)
  (finish_time_distribution 55.0 1.0)
  (quality_distribution 7.0 1.0)
  (cost_distribution 6.0 1.0)
  (duration_distribution 55.0 1.0)
  (rating 0.5897435897435898)
  )
-->
<schedule rating="0.5897435897435898">
```

```
<start>
  <distribution point="0.0" probability="1.0" />
</start>
<finish>
  <distribution point="55.0" probability="1.0" />
</finish>
<quality>
  <distribution point="7.0" probability="1.0" />
</quality>
<cost>
  <distribution point="6.0" probability="1.0" />
</cost>
<duration>
  <distribution point="55.0" probability="1.0" />
</duration>
<element id="Obtain-Ingredients">
  <quality>
    <distribution point="1.0" probability="1.0" />
  </quality>
  <cost>
    <distribution point="6.0" probability="1.0" />
  </cost>
  <duration>
    <distribution point="10.0" probability="1.0" />
  </duration>
  <start>
    <distribution point="0.0" probability="1.0" />
  </start>
  <finish>
    <distribution point="10.0" probability="1.0" />
  </finish>
</element>
<element id="Cook-Food">
  <quality>
    <distribution point="3.0" probability="1.0" />
  </quality>
  <cost>
    <distribution point="0.0" probability="1.0" />
  </cost>
  <duration>
    <distribution point="30.0" probability="1.0" />
  </duration>
```

```
    <start>
      <distribution point="10.0" probability="1.0" />
    </start>
    <finish>
      <distribution point="40.0" probability="1.0" />
    </finish>
  </element>
  <element id="Prepare-Food">
    <quality>
      <distribution point="3.0" probability="1.0" />
    </quality>
    <cost>
      <distribution point="0.0" probability="1.0" />
    </cost>
    <duration>
      <distribution point="15.0" probability="1.0" />
    </duration>
    <start>
      <distribution point="40.0" probability="1.0" />
    </start>
    <finish>
      <distribution point="55.0" probability="1.0" />
    </finish>
  </element>
</schedule>
<!--
  (spec_schedule
  (schedule_elements
  (Call-Restaurant
  (start_time_distribution 0.0 1.0)
  (finish_time_distribution 30.0 0.7 45.0 0.3)
  (quality_distribution 5.0 1.0)
  (cost_distribution 10.0 1.0)
  (duration_distribution 30.0 0.7 45.0 0.3)
  )
  )
  (start_time_distribution 0.0 1.0)
  (finish_time_distribution 30.0 0.7 45.0 0.3)
  (quality_distribution 5.0 1.0)
  (cost_distribution 10.0 1.0)
  (duration_distribution 30.0 0.7 45.0 0.3)
  (rating 0.2083333333333333)
```

```
    )
  -->
  <schedule rating="0.2083333333333333">
    <start>
      <distribution point="0.0" probability="1.0" />
    </start>
    <finish>
      <distribution point="30.0" probability="0.7" />
      <distribution point="45.0" probability="0.3" />
    </finish>
    <quality>
      <distribution point="5.0" probability="1.0" />
    </quality>
    <cost>
      <distribution point="10.0" probability="1.0" />
    </cost>
    <duration>
      <distribution point="30.0" probability="0.7" />
      <distribution point="45.0" probability="0.3" />
    </duration>
    <element id="Call-Restaurant">
      <quality>
        <distribution point="5.0" probability="1.0" />
      </quality>
      <cost>
        <distribution point="10.0" probability="1.0" />
      </cost>
      <duration>
        <distribution point="30.0" probability="0.7" />
        <distribution point="45.0" probability="0.3" />
      </duration>
      <start>
        <distribution point="0.0" probability="1.0" />
      </start>
      <finish>
        <distribution point="30.0" probability="0.7" />
        <distribution point="45.0" probability="0.3" />
      </finish>
    </element>
  </schedule>
</schedules>
```

# B. VIE-CDS Serialiser Examples

In appendix A we have already seen example output of the XTAEMS and TTAEMS serialisers. Here, we illustrate two additional output formats of VIE-CDS.

## B.1. Short-TAEMS

This is the Short-TAEMS output for the *Getting Dinner* example: the four lines specify the number of methods; the two methods along with their abstract probabilistic quality descriptions; and a quality description of the whole schedule (see "Short-TAEMS Serialiser", section 2.2.2).

```
2
Order-Dinner 0.0 56.0 9.2 20.0 56.0
Go-To-Restaurant 56.0 66.0 1.0 0.0 10.0
10.2 20.0 66.0
```

## B.2. Graphviz

This is Graphviz output for an extended version of the *Clean-Kitchen* example taken from the TÆMS white paper. The created text files can be processed by the Graphviz *dot* program to render graphical representations in various image formats.

```
digraph TAEMS {
  "Clean-Kitchen"[label=<<TABLE BORDER="0">
      <TR><TD><FONT POINT-SIZE="12.0">Clean-Kitchen</FONT></TD></TR>
      <TR><TD><FONT COLOR="gray" POINT-SIZE="10.0">q_sum</FONT></TD></
          TR>
    </TABLE>>];
  "Put-Away-Dishes"[label=<<TABLE BORDER="0">
```

```
      <TR><TD><FONT POINT-SIZE="12.0">Put-Away-Dishes</FONT></TD></TR>
      <TR><TD><FONT COLOR="gray" POINT-SIZE="10.0">q_exactly_one</FONT>
         </TD></TR>
    </TABLE>>];
  "Clean-Floor"[label=<<TABLE BORDER="0"><TR><TD><FONT POINT-SIZE="12.0
     ">Clean-Floor</FONT></TD></TR>
      <TR><TD><FONT COLOR="gray" POINT-SIZE="10.0">q_max</FONT></TD></
         TR>
    </TABLE>>];
  "Quick-And-Dirty"[shape=box, label=<<FONT POINT-SIZE="12.0">Quick-And
     -Dirty</FONT>>];
  "Wash-Counters"[shape=box, label=<<FONT POINT-SIZE="12.0">Wash-
     Counters</FONT>>];
  "Vacuum-Floor"[shape=box, label=<<FONT POINT-SIZE="12.0">Vacuum-Floor
     </FONT>>];
  "Wash-Floor"[shape=box, label=<<FONT POINT-SIZE="12.0">Wash-Floor</
     FONT>>];
  "Slow-And-Safe"[shape=box, label=<<FONT POINT-SIZE="12.0">Slow-And-
     Safe</FONT>>];
  "Vacuum-Cleaner"[shape=invtriangle];
  "Clean-Kitchen" -> "Clean-Floor"[arrowhead=none];
  "Clean-Kitchen" -> "Wash-Counters"[arrowhead=none];
  "Clean-Kitchen" -> "Put-Away-Dishes"[arrowhead=none];
  "Put-Away-Dishes" -> "Slow-And-Safe"[arrowhead=none];
  "Put-Away-Dishes" -> "Quick-And-Dirty"[arrowhead=none];
  "Clean-Floor" -> "Vacuum-Floor"[arrowhead=none];
  "Clean-Floor" -> "Wash-Floor"[arrowhead=none];
  "Vacuum-Floor" -> "Vacuum-Cleaner" [color=gray, label=<<FONT COLOR="
     gray" POINT-SIZE="10.0">consumes</FONT>>];
  "Wash-Floor" -> "Vacuum-Floor" [color=gray, label=<<FONT COLOR="gray"
      POINT-SIZE="10.0">hinders</FONT>>, style=dashed];
}
```

# C. VIE-CDS Command-Line Usage

VIE-CDS is distributed in a jar-file called `viecds.jar`. To launch VIE-CDS, it is necessary to have all other jar-files VIE-CDS depends on in classpath as well as the VIE-CDS jar-file. VIE-CDS relies on

- JSAP,
- serializer.jar from the Apache XML Project and
- the Java API for XML-Processing[1]. We recommend to use Xerces and Xalan as backend drivers.

After all prerequisites are met, VIE-CDS can be started with `java viecds.VieCDS`. Calling VIE-CDS with wrong or without parameters results in the following usage message:

```
Usage: java viecds.VieCDS
                [(-f|--format) <output format>] [(-i|--input) <input
                   format>] [(-o|--output) <output file>] [-c|--
                   convert] [(-s|--settings) <settings>] <taems file>
                   [(-n|--number) <schedule number>]

  [(-f|--format) <output format>]
        The output format for the schedule. Possible values are [XTAEMS
           , TTAEMS,
        SHORT, GRAPHVIZ] (default: XTAEMS)

  [(-i|--input) <input format>]
        The input format of the TAEMS structure. This will override the
            format
        deduced by the file extension. Possible values are: [XTAEMS,
           TTAEMS,
        PTAEMS]
```

[1] http://java.sun.com/webservices/jaxp/dist/1.1/docs/api/index.html, last visited May 2, 2007.

```
[(-o|--output) <output file>]
      Write result to this file.

[-c|--convert]
      Convert the TAEMS structure defined by <taems file> to the
         format
      defined by <output format>. No scheduling will be done.

[(-s|--settings) <settings>]
      Configure VIE-CDS with a Java properties file.

<taems file>
      A file containing a TAEMS structure. This may be a .ttaems, .
         ptaems or
      .xtaems file.

[(-n|--number) <schedule number>]
      The n-th schedule in the set of found schedules will be printed
         .
      Counting starts at 0. (default: -1)
```

The graphical user interface is called via `viecds.viewer.MainWindow`. At the time it does not yet support saving TÆMS structures in the XTAEMS encoding.

# Bibliography

[Austin, 1962] Austin, J. L. (1962). *How to Do Things with Words*. Oxford University Press, Oxford.

[Bellifemine et al., 2007] Bellifemine, F., Caire, G., and Greenwood, D. (2007). Developing Multi-Agent Systems with JADE. In *Wiley Series in Agent Technology*. John Wiley & Sons, Ltd.

[Bellifemine et al., 2006a] Bellifemine, F., Caire, G., Trucco, T., and Rimassa, G. (2006a). *Jade Programmer's Guide*. Telecom Italia S.p.A.

[Bellifemine et al., 2006b] Bellifemine, F., Caire, G., Trucco, T., Rimassa, G., and Mungenast, R. (2006b). *Jade Administrator's Guide*. , Telecom Italia S.p.A.

[Bellifemine et al., 2001] Bellifemine, F., Poggi, A., and Rimassa, G. (2001). JADE: A FIPA2000 Compliant Agent Development Environment. In Müller, J., André, E., Sen, S., and Frasson C, editors, *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, pages 216–217. ACM Press, New York NY USA.

[Bellifemine and Rimassa, 2001] Bellifemine, F. and Rimassa, G. (2001). Developing Multi-Agent Systems with a FIPA-Compliant Agent Framework. *Softw. Pract. Exper.*, 31(2):103–128.

[Bergenti and Poggi, 2001] Bergenti, F. and Poggi, A. (2001). A Development Toolkit to Realize Autonomous and Inter-operable Agents. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, pages 632–639. ACM Press, New York NY USA.

[Bordini R.H. et al., 2006] Bordini R.H., Braubach L., Dastani M., Seghrouchni A.E.F., Gomez-Sanz J.J. Leite J., O'Hare G., Pokahr A., and Ricci A. (2006). A Survey of Programming Languages and Platforms for Multi-Agent Systems. *Informatica, The*

*Second AgentLink III Technical Forum: Main Issues and Hot Topics in European Agent Research II*, 30(1):33–44.

[Carriero and Gelernter, 1989] Carriero, N. and Gelernter, D. (1989). Linda in context. In *Communications of the ACM*, volume 32, pages 444–458. ACM Press, New York NY USA.

[Cranefield and Purvis, 1999] Cranefield, S. and Purvis, M. (1999). UML as an Ontology Modelling Language. In Fensel, D. e. a., editor, *Proceedings of the IJCAI-99 Workshop on Intelligent Information Integration*. CEUR (Sun SITE Central Europe) Publications, Stockholm Sweden.

[Decker and Lesser, 1992] Decker, K. and Lesser, V. (1992). Generalizing the Partial Global Planning Algorithm. *International Journal on Intelligent Cooperative Information Systems*, 1(2):319–346.

[Decker and Lesser, 1995] Decker, K. and Lesser, V. (1995). Designing a Family of Coordination Algorithms. In Lesser, V. and Gasser, L., editors, *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), June 12-14, 1995, San Francisco CA USA*, pages 73–80. AAAI Press, Menlo Park CA USA.

[Denti and Omicini, 1999] Denti, E. and Omicini, A. (1999). An architecture for tuple-based coordination of multi-agent systems. In *Software: Practice and Experience*, volume 29, pages 1103–1121. John Wiley & Sons, Ltd.

[Durfee and Lesser, 1991] Durfee, E. and Lesser, V. (1991). Partial Global Planning: A Coordination Framework for Distributed Hypothesis Formation. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(5):1167–1183.

[Finin et al., 1997] Finin, T., Labrou, Y., and Mayfield, J. (1997). *Software Agents*, chapter KQML as an Agent Communication Language, pages 291–316. AAAI Press/The MIT Press, Menlo Park CA, Cambridge MA, London England.

[Foundation for Intelligent Physical Agents, 2004] Foundation for Intelligent Physical Agents (2004). FIPA Agent Management Specification. *FIPA Agent Management SC00023K*.

[Foundation for Intelligent Physical Agents (FIPA), 2002a] Foundation for Intelligent Physical Agents (FIPA) (2002a). FIPA ACL Message Structure Specification. *FIPA TC Communication SC00061G*.

[Foundation for Intelligent Physical Agents (FIPA), 2002b] Foundation for Intelligent Physical Agents (FIPA) (2002b). FIPA Request Interaction Protocol Specification. *FIPA TC Communication SC00026H.*

[Foundation for Intelligent Physical Agents (FIPA), 2002c] Foundation for Intelligent Physical Agents (FIPA) (2002c). FIPA Request Interaction Protocol Specification. *FIPA TC Communication SC00027H.*

[Hino et al., 2001] Hino, R., Izuhara, K., and Toshimichi, M. (2001). Message Exchange Method for Decentralized Scheduling. In *Proc. of the 4th IEEE International Symposium on Assembly and Task Planning*, Fukuoka, Japan.

[Horling et al., 1999] Horling, B., Lesser, V., Vincent, R., Wagner, T., Raja, A., Zhang, S., Decker, K., and Garvey, A. (1999). The TAEMS White Paper, Multi-Agent Systems Lab Technical Report 182, Department of Computer Science, University of Massachusetts at Amherst, Amherst MI USA.

[Huhns and Stephens, 1999] Huhns, M. N. and Stephens, L. M. (1999). Multiagent systems and societies of agents. In Weiss, G., editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 80–120. The MIT Press, Cambridge, MA, USA.

[Jennings, 1996] Jennings, N. R. (1996). Coordination Techniques for Distributed Artificial Intelligence,. In O'Hare, G. M. P. and Jennings, N. R., editors, *Foundations of Distributed Artificial Intelligence*, pages 187–210. Wiley, Chichester/London/New York.

[Jennings, 2000] Jennings, N. R. (2000). On Agent-Based Software Engineering. *Artificial Intelligence*, 177(2):277–296.

[Jung, 2003] Jung, B. (2003). VIE-CDS: A Modular Architecture for Criteria-Driven Scheduling. Master's thesis, Vienna University of Technology.

[Jung and Petta, 2003] Jung, B. and Petta, P. (2003). An Assessment of the TAEMS/DTC framework in the context of coordinated scheduling and directions for improvements. In D'Inverno, M. e. a., editor, *The First European Workshop on Multi-Agent Systems (EUMAS 2003), Dec. 18-19, 2003, St. Catherine's College, Oxford University*, University of Oxford.

[Jung and Petta, 2004] Jung, B. and Petta, P. (2004). Improving upon the TAEM-S/DTC framework in the context of coordinated scheduling. In Trappl, R., editor, *Cybernetics and Systems 2004*, pages 624–629. Austrian Society for Cybernetic Studies, Vienna.

[Kaihara and Fujii, 2005] Kaihara, T. and Fujii, S. (2005). A Proposal of Multi-agent Negotiation Mechanism Based on Dynamic Market Concept for Pareto Optimal Solution. In *Holonic and Multi-Agent Systems for Manufacturing*, volume 3593/2005 of *Lecture Notes in Computer Science*. Springer, Berlin/Heidelberg.

[Lesser, 1998] Lesser, V. (1998). Reflections on the Nature of Multi-Agent Coordination and Its Implications for an Agent Architecture. *Autonomous Agents and Multi-Agent Systems*, 1:89–111.

[Lesser and Corkill, 1983] Lesser, V. and Corkill, D. (1983). The Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks. *AI Magazine*, 4(3):15–33.

[Lesser et al., 2004] Lesser, V., Decker, K., Wagner, T., Carver, N., Garvey, A., Horling, B., Neiman, D. Podorozhny, R., Prasad, M., Raja, A., Vincent, R., Xuan, P., and Zhang, X. (2004). Evolution of the GPGP Domain-Independent Coordination Framework. *Autonomous Agents and Multi-Agent Systems*, 9(1-2):87–143.

[Logie et al., 2003] Logie, S., Sabaz, D., and Gruver, W. A. (2003). Combinatorial Sliding Window Scheduling for Distributed Systems. In *Proc. of the 2003 IEEE International converence on Systems, Man and Cybernetics, October 5-8, 2003*, volume 1, pages 630–635. IEEE Press, Washington, DC USA.

[Logie et al., 2004] Logie, S., Sabaz, D., and Gruver, W. A. (2004). Sliding Window Distributed Combinatorial Scheduling using JADE. In *Proc. of 2004 IEEE International Conference on Systems, Man and Cybernetics, October 10-13, 2004*, The Hague, The Netherlands.

[Nwana, 1995] Nwana, H. S. (1995). Software Agents: An Overview. *Knowledge Engineering Review*, 11(2):205–244.

[Nwana et al., 1996] Nwana, H. S., Lee, L. C., and Jennings, N. R. (1996). Coordination in Software Agent Systems. *The British Telecom Technology Journal*, 14(4):79–88.

[Odell et al., 2001] Odell, J., Van Dyke Parunak, H., and Bauer, B. (2001). Representing Agent Interaction Protocols in UML. In Ciancarini, P. and Wooldridge, M., editors, *Agent-Oriented Software Engineering: First International Workshop, AOSE 2000, Limerick, Ireland, June 10, 2000. Revised Papers.*, Lecture Notes in Computer Science Volume 1957, pages 201–218. Springer-Verlag, Berlin.

[Omicini and Ossowski, 2003] Omicini, A. and Ossowski, S. (2003). *Intelligent Information Agents: The AgentLink Perspective*, volume 2586/2003 of *Lecture Notes in Computer Science*, chapter Objective versus Subjective Coordination in the Engineering of Agent Systems, pages 179–202. Springer-Verlag Berlin Heidelberg.

[Pitt and Mamdani, 1999] Pitt, J. and Mamdani, A. (1999). Some remarks on the semantics of fipa's agent communication language. *Autonomous Agents and Multi-Agent Systems*, 2(4):333–356.

[Russell and Norvig, 2003] Russell, S. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Prentice Hall Series in Artificial Intelligence. Prentice Hall, Pearson Education, Inc., Upper Saddle River, New Jersey, 2nd edition edition.

[Searle, 1970] Searle, J. R. (1970). *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press.

[Smith, 1988] Smith, R. G. (1988). The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *Distributed Artificial Intelligence*, pages 357–366.

[Viroli and Omicini, 2006] Viroli, M. and Omicini, A. (2006). Coordination as a service. *Fundamenta Informaticae*, 73(4):507–534.

[Wagner et al., 1998] Wagner, T. A., Garvey, A. J., and Lesser, V. R. (1998). Criteria Directed Task Scheduling. *Journal for Approximate Reasoning (Special Issue on Scheduling)*, 19:91–118.

[Wooldridge, 2002] Wooldridge, M. (2002). *An Introduction to Multiagent Systems*. John Wiley & Sons, Chichester, England.

[Wooldridge and Jennings, 1995] Wooldridge, M. and Jennings, N. R. (1995). Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*, 10(2):115–152.

[Zlotkin and Rosenschein, 1996] Zlotkin, G. and Rosenschein, J. (1996). Mechanisms for Automated Negotiation in State Oriented Domains. *Journal of Artificial Intelligence Research*, 5:163–238.