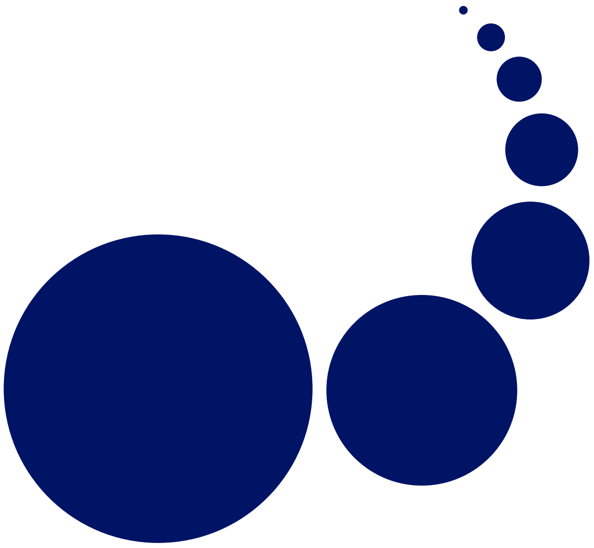
SCALABLE COMPUTING
Practice and Experience

Special Issue: Agent based systems & semantic software services

Editors: Dana Petcu and Daniela Zaharie



Volume 13, Number 1, March 2012

ISSN 1895-1767

SUBSCRIPTION INFORMATION: please visit http://www.scpe.org

Scalable Computing: Practice and Experience

Volume 13, Number 1, March 2012

TABLE OF CONTENTS

Scalable Computing: Practice and Experience
Volume 13, Number 1, pp. iii–iv. http://www.scpe.org
ISSN 1895-1767
© 2012 SCPE

INTRODUCTION TO THE SPECIAL ISSUE ON AGENT BASED SYSTEMS & SEMANTIC SOFTWARE SERVICES

Dear SCPE readers,

This issue of Scalable Computing: Practice and Experience contains papers selected and revised from those presented at three workshops:

- *3rd Workshop on Software Services: Semantic-based software services* organized as satellite workshop of FedCSIS , Szczecin, Poland, September 19-21, 2011 (`http://2011.fedcsis.org`)
- *8th Workshop on Agents for Complex Systems* and *Workshop on High Performance Computing with application in environment*, both of them organized as satellite workshops of SYNASC, Timisoara, Romania, 26-29, 2011 (`http://synasc11.info.uvt.ro`)

The six papers selected from twenty candidate papers address problems related to communication and cooperation in agent based systems, designing self-healing systems, using ontologies in designing cloud services and in enabling multi-agent systems with semantic components.

Dana Petcu,
Computer Science Department
West University of Timisoara
and Institute e-Austria Timisoara, Romania
Daniela Zaharie
Computer Science Department
West University of Timisoara

Scalable Computing: Practice and Experience
Volume 13, Number 1, pp. 5–20. http://www.scpe.org
ISSN 1895-1767
© 2012 SCPE

FASTFIX: A CONTROL THEORETIC VIEW OF SELF-HEALING FOR AUTOMATIC CORRECTIVE SOFTWARE MAINTENANCE

B. GAUDIN[*], M.H. HINCHEY[*], E. VASSEV[*], P. NIXON [†], J. COELHO GARCIA [‡] AND W. MAALEJ [§]

Abstract. One of the main objectives of self-adaptive systems is to reduce maintenance costs through automatic adaptation. Self-healing is a self-adapting property that helps systems return to a normal state after a fault or vulnerability exploit has been detected. The problem is intuitively appealing as a way to automate the different type of maintenance processes (corrective, adaptive and perfective) and forms an interesting area of research that has inspired many initiatives. As a result, several surveys on self-healing have been published to describe the state of the art in this field. According to those surveys, the major trend towards finding a solution of the self-healing problem relies on redundancy that may concern both architecture and code resources. These approaches are therefore better suited to address adaptive and perfective maintenance. As part of the EU FP7 FastFix project [1], we focus on self-healing for corrective maintenance. We propose a framework for automating corrective maintenance that is based on software control principles. Our approach automates the engineering of self-healing systems as it does not require the system to be designed in a specific way. Instead it can be applied to legacy systems and automatically equip them with observation and control points. Moreover, the proposed approach relies on a sound control theory developed for Discrete Event Systems. Finally, this paper contributes to the field by introducing challenges to the effective application of this approach to relevant industrial systems. Some of these challenges are currently being tackled within FastFix.

Key words: software maintenance, self-healing, software control, context-aware software engineering

1. Introduction.

Software maintenance aims to modify software systems after they are deployed in production ([39, 14]). In [47], the authors divide maintenance activities in three different types: adaptive, perfective, and corrective. Adaptive maintenance is performed to make the computer program usable in a changed environment. Perfective maintenance mainly tackles performance and maintainability issues. Corrective maintenance is performed to correct faults. Within the last 20 years the complexity of both software and communication infrastructures has increased at an unparalleled rate. This level of complexity means that software systems are more prone to unexplained faults, require more support and maintenance, and cost more to deploy and manage. A fundamental challenge faced by the software industry is how to ensure that these complex systems require less maintenance and human intervention. Concepts such as self-healing, autonomic and self-adaptive systems provide an answer by reducing human intervention and reducing the apparent complexity of systems.

Several surveys on self-healing have been published to describe the state of the art of this field. According to these surveys, the major trend towards finding a solution of the self-healing problem rely on redundancy that may concern both architecture and code resources. These approaches assume that systems are designed with adaptive capabilities and are therefore better suited to address adaptive and perfective maintenance. In this article, we focus on self-healing for corrective maintenance. Section 2 recalls existing works on self-healing, automatic diagnosis, and automatic repair of software systems.

We also propose a control theoretic approach to self-healing in order to deal with corrective maintenance. Control makes it possible to drive the system in a range of desired behaviours. It represents an interesting approach to avoid behaviours leading to failures. This is achieved by dynamically disabling some of the implemented features, depending on the current execution of the system. Moreover, the proposed approach automatically synthesizes supervisors in charge of controlling the software. This hence automates the computation of a new suitable range of software behaviours whenever corrective maintenance needs to be performed, e.g. a failure has been reported and behaviours leading to this failure need to be removed or avoided. Our approach consists of a pre-deployment and runtime phase. Each phase is described in Sect. 3. Sect 4 illustrates our approach through one of the case studies considered in FastFix: the Moskitt application.

Finally, challenges to be tackled in order to implement effective and efficient control theoretic self-healing features are discussed in Sect. 5. Most of these challenges relate to the supervisory control theory and its applicability to software system. However we also show that some challenges are common to the research in automatic diagnosis and automatic repair.

[*]Lero - The Irish Software Engineering Research Center, Limerick, Ireland ({benoit.gaudin, emil.vassev@lero.ie, mike.hinchey@lero.ie}@lero.ie).
[†]University of Tasmania, Hobart, Australia (Paddy.Nixon@utas.edu.au)
[‡]Instituto de Engenharia de Sistemas e Computadores Investigação e Desenvolvimento em Lisboa, Lisbon, Portugal (jog@gsd.inesc-id.pt)
[§]Technische Universität München, Munich, Germany (maalejw@cs.tum.edu)

2. Overview of Automated Software Maintenance Approaches.

2.1. Self-Healing. Self-healing is a concept which aims to tackle or prevent maintenance tasks in presence of system failures. This concept came with the notion of Autonomic Computing initiated by IBM in [32]. The main goal of Autonomic Computing is to reduce human intervention in component management.

There are different visions of self-healing in the literature. In [52], Rodosek et al. consider self-healing as equivalent to self-repairing and self-immunity, i.e. the ability to resist to infections. A self-healing system must be able to recover and go back to a proper state following some disturbance. This view is shared in [26], where recovery oriented computing is presented as a key aspect of self-healing. The authors consider that healing systems are more concerned with post-fault or post-attack states and more specifically with bringing the system back to a normative state. In [34], the author has a broader view of self-healing. In that work, self-healing tries to identify and eliminate or mitigate the root cause of the fault. In [37], the authors have a similar view of self-healing and recall that the system requires knowledge about its expected behaviours in order to automatically discover system malfunctions or possible futures failures. In [53] the authors describe self-healing as consisting of self-diagnosis and self-repair. The consequence of this observation is that work related to automatic diagnosis and automatic repair or recovery are relevant to the field of self-healing. Finally although self-healing aims for automation, as discussed in works such as [26, 50], even non fully automated healing approaches may also represent self-healing techniques. In [26] for instance, Ghosh et al. introduce the notion of assisted-healing for systems that require some human intervention during their healing process.

Historically, self-healing techniques were inspired by fault-tolerance and these two fields are tightly connected as explained in [15]. This entails that, as for fault-tolerance approaches, self-healing solutions often rely on some system redundancy, such as components, hardware, network nodes, code variants, etc. This observation was also made recently in [45] which also provides a survey on self-healing approaches. The authors classify self-healing techniques and faults tackled. Self-healing techniques can be classified according to the type of systems under consideration and span over service relaunch, checkpointing, architecture based, model based, multi-agent based, reflection based, aspect oriented programming, service discovery and load balancing. From [45] again, the faults tackled by self-healing approaches are classified into crash failure, fail-stop (execution is deliberately inhibited on a failure and detected by other processes), omission (message loss or transmission error), transient (error related to presence of various self recovering faults disturbing other parts of the system), timing and performance (constrained distributed synchronous execution of tasks by a specific amount of time), security and arbitrary (a process confuses the neighbors by providing constantly individual consistent but contradicting information). As pointed out in [55], self-healing is still a relatively immature field and the class of faults tackled by this field remains quite narrow. Moreover, the techniques employed in this field mainly use architecture adaptation in order for the system to provide expected features even without the faulty component. To this respect, these approaches achieve adaptive maintenance in presence of failures.

Recently, other self-healing approaches modifying the behaviours in order to correct them have been considered, e.g. [56, 7, 8, 6]. Therefore these approaches propose self-healing features that are suitable for corrective maintenance.

In [56], the authors present ASSURE, a self-healing approach based on rollback and error handling facilities. When an error occurs at runtime and the system is brought back to a rescue point, pre-defined error handling strategies are executed in a virtualized environment and tested. If it is satisfactory, then the error handling code is applied to production code, modifying the initial system behaviour in order to correct it. This approach makes it possible to self-heal from unknown issues by applying recovery approaches for known issues, that also seem to apply to the unknown ones.

Carzaniga et al. [7, 8, 6] consider a self-healing approach that modifies the behaviour of the component to be healed. The proposed approach, called workaround, consists of replacing a faulty sequence of operations with another that produces the same outputs or effects. Workaround is a model based approach which provides alternative program executions to the failing ones. This approach relies on the observation that libraries often contain feature redundancies. A typical example, provided in [6] is the one of changing an item in a shopping basket. The change item feature can be achieved by composing the remove item and the add item features, i.e. an item change can be seen as the removal of an item followed by the addition of another one.

As explained in [53] self-healing can be seen as a combination of self-diagnosis and self-repair approaches. A broader view of these concepts are *automatic diagnosis* and *automatic repair* which are strongly related to the corrective maintenance process.

2.2. Automatic Diagnosis. Diagnosis is a *proactive* software-maintenance technique driven by *detection* and *isolation* of faults to prevent failures [47]. *Automatic diagnosis* targets the automation of the diagnosis process, where faults are detected and isolated by the system itself, often by applying techniques working on the system architecture or by implementing special *alarms*. The *architecture-based techniques* usually rely on resource redundancy. For example, in [54] is considered the feasibility for a multi-processor system to perform self-diagnosis on some of its processors by some others. Another class of automatic diagnosis techniques is the so-called *correlation-based diagnosis* that considers diagnosis that may provide several types of alarms where an issue is detected by the raise of one or more alarms. In [16] correlation-based diagnosis is applied to Discrete Event Systems and considers the detection of alarms whenever they are not directly observable. In another approach metrics related to system states or performance are correlated as a means for diagnosis [31]. Normal system behaviour is determined by specific metric correlations and faults might be detected whenever there is a deviation from these metrics correlations. Finally, model based techniques form another class of automatic diagnosis. System models take as input some observations of the current system state or behaviour and produce diagnosis. In general, the model based diagnosis is about comparing a system behaviour with actual observed executions [51]. When the observed execution deviates from the expected behaviour provided by the model, this is an indication of a fault occurrence. [29] consider probabilistic models in order to apply this principle.

2.3. Automatic Software Repair and Bug Fixing. Several approaches have been proposed to automate the bug fixing process. *Rollback techniques* maintain a record of "healthy" system states to allow a rollback to the last such state when a fault occurs. Once successfully rolled back to a *healthy state*, the system re-executes after applying certain changes to its input data or execution environment (see e.g. [46, 56, 60, 35, 5]). *Mutation techniques* rely on *Genetic Programming* concepts and are closely related to data structure linking and modification. The *data structure repair* approach [42, 17, 18, 21] uses structural integrity constraints for key data structures to monitor their correctness during the execution of a program. If a constraint violation is detected, then mutations are performed on the system data structures in order to transform them so that they satisfy the desired constraints. *Event Filtering techniques* are usually related to software *security* and *vulnerability*. They consist of automatically creating and detecting *signatures* or *patterns for malicious* attacks such as *control hijacking* and *code injection*. Then these signatures are used for a *filtering check*, so that such attacks cannot break through in the system anymore. Systems following this principles are *PASAN* ([59]), *FLIPS* ([41]) and *ShieldGen* ([13]). *Learning* and *probabilistic* approaches to automatic repair and bug fixing learn from past executions where bugs have been fixed. Applied fixes are stored and can be retrieved and applied again or used in order to infer other possible fixes. Systems such as *Exterminator* ([43]), *BugFix* ([30]) and *ClearView* ([44]) implement such a principle. Finally, in [61] the authors present AutoFix-E, an automatic code fixing approach based *Model Checking*. This approach considers contract violations as failures and calls existing functions whose postcondition fulfills the violated contract. Fix candidates are created from a set of fix templates and the behaviour models.

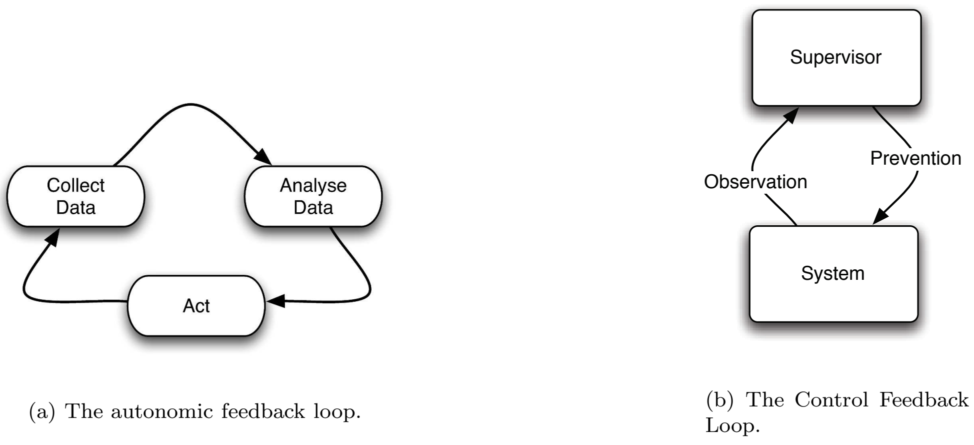
2.4. Conclusion. Self-healing approaches mainly rely on system redundancy which adapt their architecture in order to bypass faulty components but still provide their expected features. Therefore these approaches are related to adaptive maintenance, where the system adapts to changes due to failures.

The authors of [53] suggest a definition of self-healing consisting of self-diagnosis and self-repair. As diagnosis and repair techniques are very relevant to corrective maintenance, such a vision of self-healing is well suited to this maintenance task. However, automatic diagnosis and repair techniques found in the literature focus on analyses and lack of a unified and systematic approach for equipping the system with self-healing facilities implementing the autonomic feedback loop (see Figure 3.1(a)).

In Sect. 3 we propose an approach for software self-healing that automatically introduces autonomic facilities into an existing system, e.g. sensors and actuators. This approach is based on The Supervisory Control Theory (SCT) for Discrete Event Systems where the corrective maintenance task corresponds to the automatic synthesis of a supervisor. Section 5 introduces the main research challenges associated to the proposed approach.

3. A Control Theoretic Approach to Software Self-Healing. Regarding computing systems, control theory has traditionally been applied to data networks, operating systems, middleware, multimedia and power management ([28]). This section introduces a control-based approach for software self-healing.

Self-Healing is a property of Autonomic Systems [33]. Our approach proposes to automatically equip software systems with autonomic features before deployment so that they can follow the different phases of



(a) The autonomic feedback loop.

(b) The Control Feedback Loop.

Fig. 3.1: The runtime autonomic and control feedback loops.

the autonomic feedback loop presented in Figure 3.1(a). In particular, sensors and actuators are automatically added to the software system in order to realize the *Data Collection* and *Action* phases of Figure 3.1(a). Within FastFix, the *Analysis* phase corresponds to automatic supervisor synthesis. As explained in [20] control theory principles are suitable to implement the autonomic feedback loop. More specifically, this work considers the Supervisory Control Theory (SCT) on Discrete Event Systems. This theory was initiated in [48] and is a model based approach aiming to automate the synthesis of correct models.

Our self-healing approach consists of two different parts: a pre-deployment part which is performed before the system is deployed and where self-healing features are added to the software; and a post-deployment part corresponding to the automatic or semi-automatic execution of the maintenance process where the system self-healing features are employed. The latter part itself consists of supervisor synthesis and runtime supervision. Synthesis is applied using SCT, whenever new runtime system specifications need to be ensured, e.g. when a fault has occurred and behaviours leading to it must be removed. Runtime supervision corresponds to applying the synthesized supervisor to the application at runtime. Overall the presented approach can be seen as a three phase approach: pre-deployment, supervisor synthesis and runtime supervision. These phases are presented in more details in Sect. 3.1.

3.1. Overall Approach. The overall proposed approach is depicted in Figure 3.2. The left-hand side of this diagram represents the pre-deployment phase during which code is instrumented in order to introduce observation and control points (i.e. sensors and actuators) as well as data structures that make it possible for the application to embed and use supervisor models. A binary (or bytecode) application with these facilities can then be obtained through compilation. During the pre-deployment phase, a model of the behaviours is also automatically extracted from source code through control flows and method calls analysis.

During the runtime and maintenance phase, the software artefacts (source code or bytecode) are no further modified. Only models of a supervisor representing their possible runtime behaviours are manipulated in order to maintain the application behaviours within a desired set. These models are embedded in the application at runtime and are modified and replaced whenever an error occurs so that behaviours leading to this error cannot occur in future system executions.

Some unknown possible failures of the system may occur at runtime, requiring the application to be corrected. The observation of such a failure indeed indicates that the system behaviour is not satisfactory and needs to be modified. Self-healing capabilities aim to correct the system behaviour so that the observed failure can no longer occur. Such corrections are performed by modifying the supervisor that interacts with the application at runtime. Considering the Supervisory Control Theory introduced in [48], this can be automatically achieved when a control objective is provided. In some situations, this control objective can be automatically derived from observations of failures during the system execution [25]. In general, control objectives can also

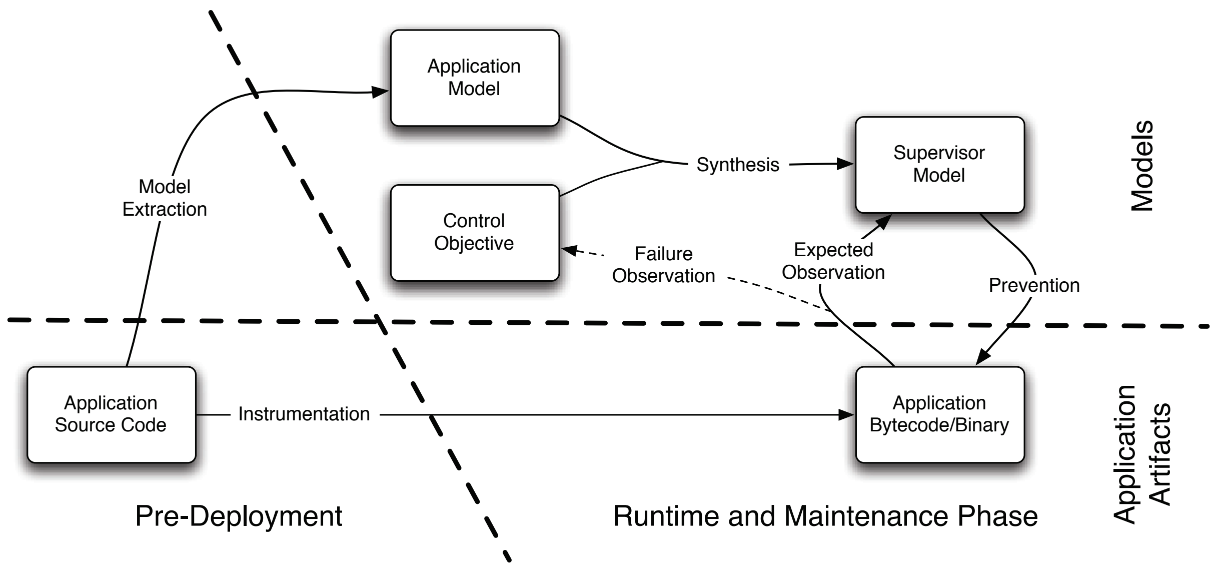


Fig. 3.2: A Software Control Approach to Self-Healing.

be provided by expertise. The accuracy and relevance of the expertise involved in designing a control objective will impact on the accuracy and relevance of the corrective solution applied to the system. For instance, diagnosis can help design a more accurate control objective. However, in cases where deep analyses and diagnostics cannot be conducted (e.g. when the amount of time that is necessary to perform this task is too long), a simple control objective excluding the undesired previously observed sequences of method calls can be submitted to the supervisor synthesis algorithm. However in this case, the resulting supervisor may act more coarsely and unnecessarily remove some of the system behaviours. This depends on how representative of an undesired behaviour the observed sequence is.

3.2. Pre-Deployment Phase. The pre-deployment phase aims to prepare the software application so that control and synthesis can be performed at runtime. This preparation consists of 2 subtasks: code instrumentation and model extraction. Each of these tasks is performed in an automated fashion.

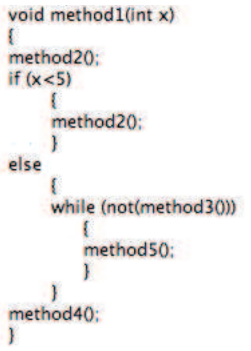
Code instrumentation is performed in order to introduce observation and control points as well as to embed a supervisor in the application, as illustrated in Figure 3.6. Intuitively, automatically instrumenting the application code consists of automatically embedding a supervisor into the system as well as adding conditional statements in each method body so that method invocations can be observed and executions of method bodies can be prevented at runtime [1].

Moreover the approach introduced in this section relies on the automatic design of a model of the application behaviours. In its basic form, this model can be a Finite State Machine whose transitions represent method calls. An over-approximation of the behaviours of the application can be obtained from the source code by considering methods, branching and loops as illustrated in Figure 3.3.

Some tools have been implemented in order to extract and analyze models represented as Extended Finite State Machines (EFSM), i.e. FSM associated with variables. PROMELA is an FSM-based modeling language. PROMELA models can be used as input to the SPIN tool, which can then model-check this model against some properties. Bandera ([11]) extracts FSM from Java code. Bandera offers the possibility of exporting the extracted models into the PROMELA format. More recently in [27], the authors proposed an efficient approach for model extraction from programs. The approach makes it possible to deal with different but syntactically similar programming languages such as C++ and Java.

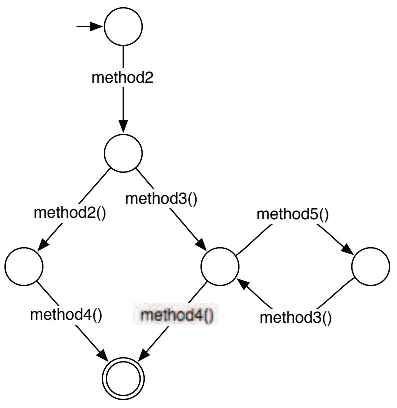
In all these approaches however, only some particular parts of the programming language are considered. When the extracted models are meant to be used for model-checking, the choice of the program parts to be

[1]More details on the runtime aspect is provided in Sect. 3.4.



```
void method1(int x)
{
method2();
if (x<5)
     {
     method2();
     }
else
     {
     while (not(method3()))
          {
          method5();
          }
     }
method4();
}
```

(a) An example of method declaration.



(b) An FSM modeling the system behaviours of method1.

Fig. 3.3: Illustration of FSM extraction.

extracted can be driven by the property to be model-checked (e.g. [11]). In case of software maintenance, one usually does not know which part of the application is faulty before an error occurs. Therefore monitoring relevant information about the occurrence of an error requires to cover a large part of the application. Moreover as the relevance of approach for self-healing relies on the observation made at runtime, it requires that the application models encode these possible runtime observations. In our approach, the extracted models actually encode all the possible occurrences of method invocations, for methods declared in the application, i.e. invocations of methods declared in external components are not considered. This characteristic is related to the fact that the extracted model is used for on-line monitoring and capture relevant information when an error occurs. Therefore an important challenge for model extraction consists of obtaining a complete application model. This requires that the model complies with the specification of the language compiler or virtual machine so that features such as threads and graphical components are treated appropriately.

In order to extract models on large applications, we use a modular approach. A typical output of the model extraction mechanism is depicted in Figure 3.4. It consists of a set of Finite State Machines, each of them possessing one initial state (represented as an hexagon in the figure) and possibly several final states (represented as double-circle states). From each of these initial state, only one event can be triggered, i.e. event m_i for each FSM$_i$ and for $1 \leq i \leq 3$. Moreover these events, called *triggering events*, do not appear in any other transition or in any other FSM. Therefore, when observed at runtime, these events uniquely characterize which FSM is running and initiates any of the behaviours of this FSM.

Considering software applications, triggering events represent methods that are not called from within the application. In our approach, this also takes into account the fact that methods within the application may call an external method that is overridden by a method that is declared in the application itself. This means that triggering events may only be called through external events such as a call from an external component, from user interactions, from occurrences of system events, etc.

Triggering events make it possible to capture concepts such as the behaviours associated to button clicks of a graphical interface (e.g. method *actionPerformed* in Java SWING), the start of a new thread (e.g. method *run* in Java), etc.

Run methods represent concurrency in the application at runtime. This concurrency is also present in the model as triggering events are declared in different FSM that can run concurrently. However more modularity is also introduced in the model whenever this is possible. For instance, a method may be a triggering event although its behaviour does not run concurrently. For instance all actionPerformed methods run on the same thread, the *EventDispatch* thread. In this case, apparent concurrency in the model does not represent actual concurrency at runtime. This approximation is however an interesting means to lower the complexity of the

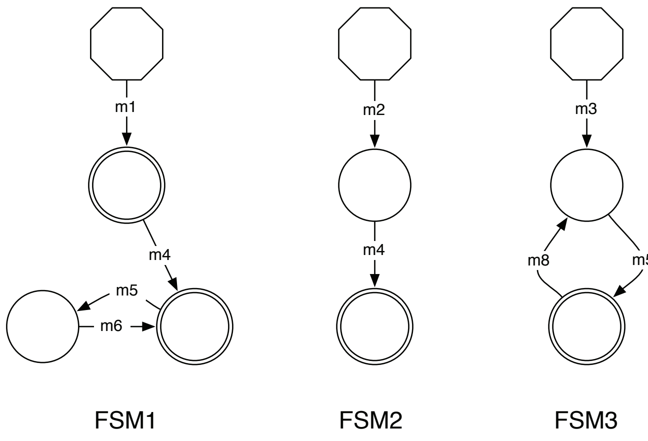


Fig. 3.4: Structure of the extracted FSMs.

extracted models. A modular model extraction approach indeed avoids the computation of a single Finite State Machine, which would become intractable for large applications. This approach allows to extract models of all the possible application behaviours and makes it scalable by splitting the problem into the one of extracting an FSM for each of its entry point.

The runtime dynamic of this model is described in more details in Sect. 3.4.

3.3. Synthesis Phase. The design of such a supervisor corresponds to determining how the application behaviours must be modified in order to avoid undesired behaviours. However designing such a supervisor is a challenging and prone to error task. Moreover the high complexity of software applications makes it difficult to take manually into account all the possible failures that can occur and need to be prevented. For this reason, supervisors may need to adapt at runtime so that they take into account newly observed undesired behaviours, hence performing corrective maintenance. Such an approach is described in Figure 3.5(a).

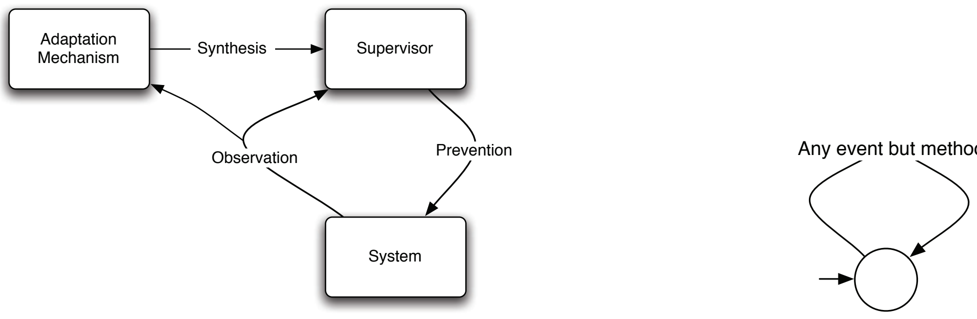
Our approach considers the automatic synthesis of such supervisors. More specifically we consider techniques that automatically compute the model of a supervisor given a model of the application behaviour and a model representing a set of desired behaviours[2]. The Supervisory Control Theory (SCT) on Discrete Event Systems introduced by Ramadge and Wonham [49], offers such a framework and techniques for the automatic synthesis of supervisors.

SCT is a formal theory that aims to automatically design a model for a supervisor ensuring some safety property. The Supervisory Control Theory defines notions and techniques that allow for existence and automatic computation of a model of the supervisor, given a model of the system as well as the property to be ensured. In this theory, models of a system G are represented by languages over alphabets of events, denoted $L(G)$. These languages correspond to sets of sequences of events, each representing a possible behaviour/execution of the system.

Although not as general as languages, Finite State Machine (FSM) are used to model the possible behaviours of the system as well as the supervisor and the properties to be ensured by control. Regarding the modeling of supervisors, Figure 3.1(b) shows that they can be seen as a function that takes a given sequence s and returns to the system a set of allowed events after s. The function S representing the supervisor can be encoded by a FSM G_S such that for all $s \in L(\mathrm{S})$, $S(s)$ represents the set of events that can be triggered from the state reached in G_S after sequence s.

Supervisors ensure a given property, called *control objective*. Such a property is modeled as a FSM as well, generating a set of "safe" behaviours and meaning that the behaviours that are not encoded by this FSM are undesired. For instance, Figure 3.5(b) represents a very simple control objective which models that method1 must never be executed.

[2]Behaviours that do not belong to this set are undesired.



(a) A Software Control Adaptation View.

(b) A simple control objective.

Fig. 3.5: Software control Adaptation view and a simple control objective.

The main goal of the Supervisory Control theory is to automatically synthesize a model of a supervisor that ensures that the system behaviours are all included in the ones described by the control objective. The theory also considers that not every event can or should be disabled by a supervisor. Such events are said to be uncontrollable. In order to take such events into account, the alphabet of the system is assumed to be composed of a set of *controllable* events ($A_c \subseteq A$) and *uncontrollable* events ($A_u \subseteq A$). Each event of the system is either controllable or uncontrollable. Controlling a system consists of restricting its possible behaviours taking into account the controllable nature of the system events. In order to achieve this, Ramadge and Wonham (see e.g. [63]) introduce a property called *Controllability.* A system G' whose behaviours correspond to a subset of the ones of G is controllable w.r.t A_u and G if $L(G').A_u \cap L(G) \subseteq L(G')$. A controllable set of behaviours G' ensures that no sequence of uncontrollable events can complete a sequence of G' into a sequence of G that is no longer in G'. In other words, the controllability condition ensures the synthesized supervisor can be effectively implemented with respect to the available controllable events. We now define the basic supervisory control problem, which can be stated as the following:

Basic Supervisory Control Problem (BSCP): Given a system G and a control objective K, compute the maximal controllable set of behaviours included in the ones of both G and K.

Ramadge and Wonham (see e.g. [63]) have shown that a solution to the BSCP exists if and only if the maximal controllable set of behaviours included in the ones of both G and K is not empty. They also provided an algorithm computing this FSM which encodes a most permissive supervisor ensuring the control objective (see e.g. [63]). This algorithm can be seen as a function that takes as inputs a set of uncontrollable events A_u, a FSM representing the control objective K and a FSM representing the behaviours of the system G. In our proposed approach, corrective maintenance is applied by modifying the application behaviours. Determining the set of behaviours to be ensured by control is performed solving the BSCP. The obtained model is then use to control the application. Part of the mechanism involved to achieve this is described in Sect. 3.4 and part of it is performed during the pre-deployment phase and is described in Sect. 3.2.

The control objective of Figure 3.5(b) illustrates the case where it is desired to prevent occurrences of method1. Although in some situations such an objective represents the most relevant property to ensure on the system, it may also represent an approximation due to lack of knowledge. The root cause of the failure that leads to the design of this control objective may not indeed come from method1 but from other methods calling method1. If the developers can only observe that the failure occurs when method1 is executed, then preventing the occurrence of method1 appears to be the most straightforward way to avoid the failure.

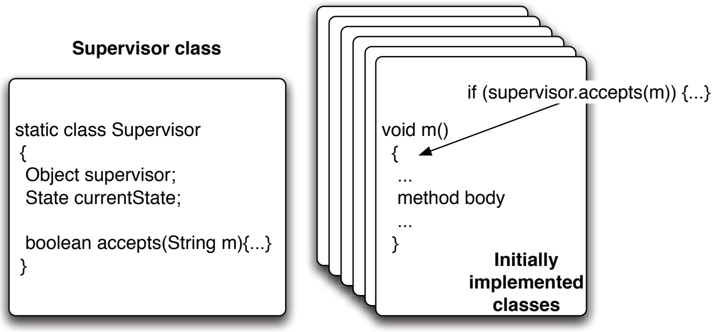
The algorithm solving the BSCP provides a new model of a supervisor which will be used by the application in order to prevent the future occurrence of undesired behaviours. In general, a restart of the application is necessary in order to take into account the newly computed supervisor model.

Finally the extracted models are represented as a composition of FSMs. Classical supervisory control techniques require that a single FSM represents the system behaviours. Such a FSM can be obtained by computing the composition of the FSM representing each component. However, this computation leads to a state explosion problem and represents an important challenge of the supervisory control theory. Some works on control on concurrent systems have been conducted (e.g. [62, 19, 24]) and can be applied to the extracted model. In particular, conditions stated in [24] for efficient modular supervisor synthesis are fulfilled by the model extracted as in Sect. 3.2. For instance one such requirement is that shared event between FSMs are controllable. This requirement always hold with our model at runtime as there is actually no shared event between the modelled concurrent FSMs. This is due to the fact that each FSM is being executed on a different thread at a given time and that the knowledge of the thread on which a method is invoked indicates which FSM this event is belonging to.

3.4. Runtime Supervision. When an error occurs at runtime, the observed behaviour is used in order to modify the extracted model as described in Sect. 3.3. The resulting model encodes a supervisor to be applied to the application at runtime. This section describes this mechanism.

We first consider the control phase which follows the principle illustrated in Figure 3.1(b). In this diagram, the supervisor observes and controls the current behaviours of the system. These behaviours are represented as sequences of events.

As illustrated in Figure 3.6, the model of the supervisor is embedded in the application. More specifically, the model of the supervisor can be considered as an object whose current state can be updated whenever a method of the application to be controlled is invoked. Each time a method is called, then method *accept* is called. First, this method makes the supervisor aware of the method being invoked and updates its knowledge of the current behaviour of the application. Second, this method returns a boolean value indicating whether the supervisor allows the body of the method to be executed. Such an approach allows for dynamic restriction of the system executions, e.g. a method execution may be prevented after a given sequence and allowed after another one.


Supervisor class
Initially implemented classes
if (supervisor.accepts(m)) {...}

```
static class Supervisor
{
 Object supervisor;
 State currentState;

 boolean accepts(String m){...}
}
```

```
void m()
 {
  ...
  method body
  ...
 }
```

Fig. 3.6: A possible code instrumentation offering observation and control points.

The models obtained from model extraction and presented in Figure 3.4 represent concurrent Finite State Machines. However the concurrency between these FSMs may not correspond to the one of the threads created during the execution of the application. Considering Figure 3.4 again, although FSM$_1$ and FSM$_2$ are modelled as concurrent FSMs, it may be the case that m_1 and m_2 are always executed on the same thread. This may happen for instance when the application to be controlled is an API and m_1 and m_2 are always called from methods of an external component that run on the same thread.

The dynamic of the concurrent FSMs of the model we consider is unlike the standard parallel composition of FSMs (see e.g. [9]). Instead the dynamic of the model considers that only a subset of the concurrent FSMs may run simultaneously. This mechanism is embedded in the implementation of the supervisor and consists of

- mapping at runtime the observed current thread and method call to the appropriate running FSM in order for it to update its current state,
- mapping at runtime the observation of a triggering event, i.e. the first event that can be triggered from a FSM to the corresponding FSM.

Figure 3.7 illustrates the runtime mechanism of concurrent FSM model. We assume here that the model consists of a pool of n concurrent FSMs $\{G_0, \ldots, G_n\}$. In this example, the first method invocation observed is the triggering event associated to G_3 and is executed on thread *Thread1*. Then some of G_3's behaviours may be executed on this thread as well as the triggering event of FSM G_6 on thread *Thread2*. Then G_3 and G_6 run in parallel on their respective threads when the method corresponding to the triggering event of G_2 is invoked on thread *Thread3*. Then the current behaviours of G_3 completes and the method associated to the triggering event of G_7 is invoked on thread *Thread1*. Finally, FSMs G_7, G_6 and G_2 run in parallel on their respective thread until the behaviour of G_6 completes.

This runtime dynamic is sound as triggering events only occur from the initial state of an FSM and do not appear in any other ones. Therefore when a method corresponding to a triggering event is invoked on a thread, there is no ambiguity as to whether it initiates the behaviour of an FSM on this thread or extends the behaviour of the FSM currently associated to this thread: the first case indeed applies. Moreover when a method that does not correspond to a triggering event is invoked on a given thread, it corresponds to the a transition of the FSM currently associated to this thread. The information about the thread on which the method is called removes any ambiguity on the FSM for which the corresponding event belongs to.

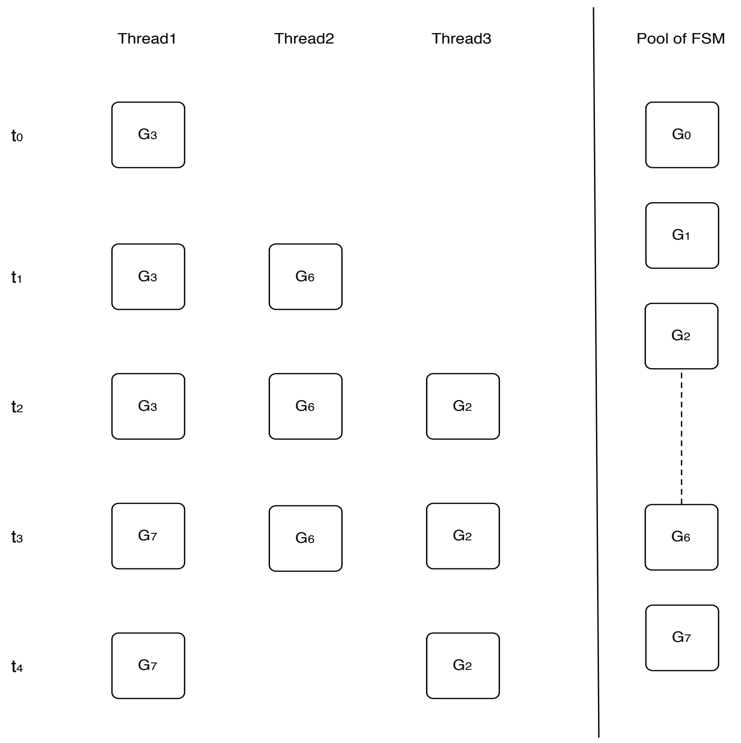


Fig. 3.7: The Runtime Dynamic of the Model Concurrent FSMs

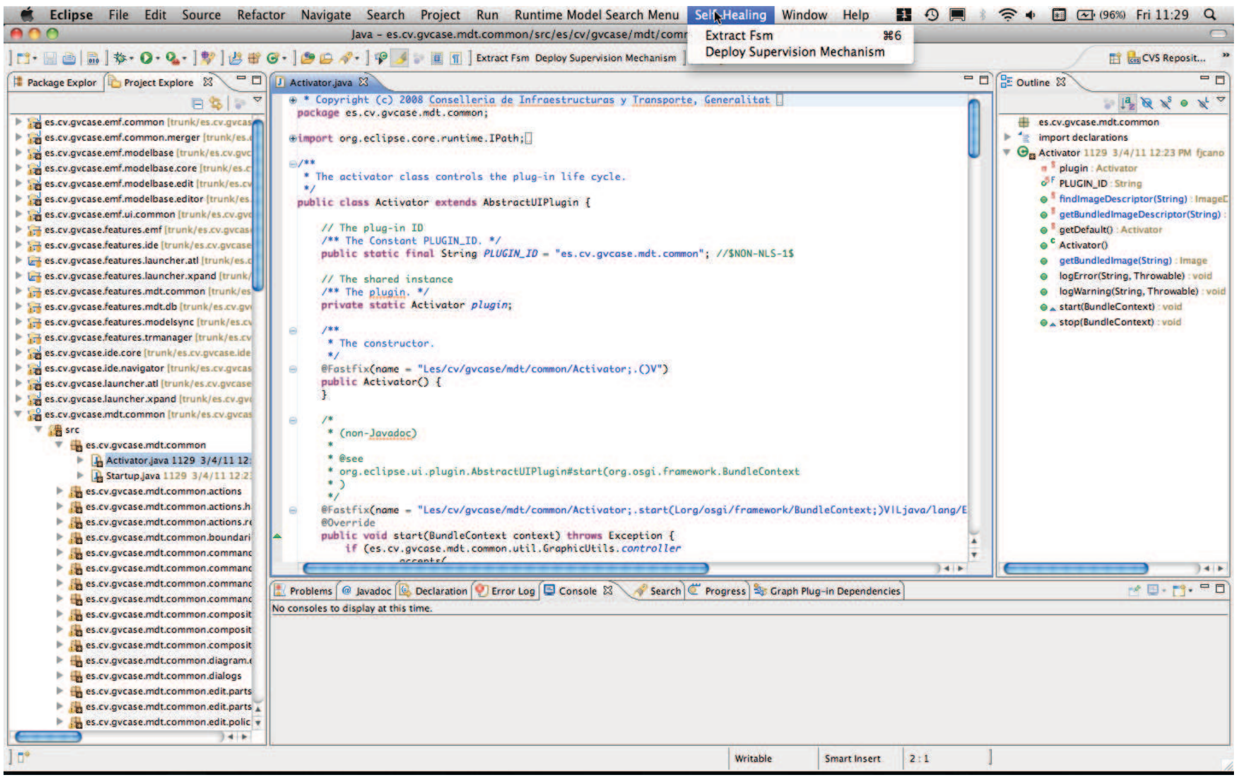
Finally, the supervisor embedded in the FastFix target application is a declared as a *synchronized* object and it is therefore safe to call it form different threads. Such an approach makes it possible for the supervisor to control behaviours that spread over several threads. However, this approach introduces some extra concurrency between threads, i.e. threads have to share an extra resource: the supervisor.

3.5. Summary. The control theoretic approach for self-healing proposed in this section raises several challenges. Some of these challenges correspond for instance to automating the introduction of autonomic features into legacy applications; automatically extracting relevant and accurate models from source code; applying supervisory control theory on large systems; designing accurate control objective, etc. They also relate to different fields of computer science such as software engineering (e.g. software modeling, logging, maintenance), formal methods and control theory. Sect. 4 illustrates this approach on an industrially relevant application: Mokistt while Sect. 5 presents challenges related to our approach.

4. Example. This section applies the approach described in Sect. 3. More specifically, it illustrates the pre-deployment phase on a industrially relevant application: Moskitt [2]. Moskitt is an open source software initially developed for the *Conselleria de Infraestructuras, Territorio y Medio Ambiente*, built on top of Eclipse and which supports modeling tasks. This application is used as a case study within the FastFix project. It consists of numerous modules implemented as OSGI bundles [3]. The applicability of our automated model extraction and supervision deployment mechanisms is illustrated on Moskitt.

Our model extraction and supervision deployment mechanisms have been implemented as an Eclipse plugin, illustrated in Figure 4.1. Table 4.1 presents results regarding the scalability and efficiency of the approach and Figure 4.2 illustrates the outcome of the instrumentation embedding supervisors within the application.

As shown in Figure 4.1, our plugin implements the pre-deployment phase of our self-healing approach, and contains two features: model extraction and supervision deployment. Model extraction is performed through static analysis of the application source code. The different Moskitt bundles appear on the left-hand side of Figure 4.1. For this example, we used a MacBook Pro with a 2.6Ghz dual core i7 processor and 4GB of RAM.


Eclipse File Edit Source Refactor Navigate Search Project Run Runtime Model Search Menu Self-Healing Window Help
Extract Fsm ⌘6
Deploy Supervision Mechanism
Java - es.cv.gvcase.mdt.common/src/es/cv/gvcase/mdt/comr
Extract Fsm Deploy Supervision Mechanism
Package Explor
Project Explore
Activator.java
Outline
Problems
Javadoc
Declaration
Error Log
Console
Search
Progress
Graph Plug-in Dependencies
Writable
Smart Insert
2 : 1

```
 * Copyright (c) 2008 Conselleria de Infraestructuras y Transporte, Generalitat
package es.cv.gvcase.mdt.common;

import org.eclipse.core.runtime.IPath;

/**
 * The activator class controls the plug-in life cycle.
 */
public class Activator extends AbstractUIPlugin {

    // The plug-in ID
    /** The Constant PLUGIN_ID. */
    public static final String PLUGIN_ID = "es.cv.gvcase.mdt.common"; //$NON-NLS-1$

    // The shared instance
    /** The plugin. */
    private static Activator plugin;

    /**
     * The constructor.
     */
    @Fastfix(name = "Les/cv/gvcase/mdt/common/Activator;.()V")
    public Activator() {
    }

    /*
     * (non-Javadoc)
     *
     * @see
     * org.eclipse.ui.plugin.AbstractUIPlugin#start(org.osgi.framework.BundleContext
     * )
     */
    @Fastfix(name = "Les/cv/gvcase/mdt/common/Activator;.start(Lorg/osgi/framework/BundleContext;)V|Ljava/lang/E
    @Override
    public void start(BundleContext context) throws Exception {
        if (es.cv.gvcase.mdt.common.util.GraphicUtils.controller
```

```
No consoles to display at this time.
```

Fig. 4.1: Screenshot of the FastFix Self-Healing Component applied to Moskitt.

Table 4.1 presents results about the model extraction and supervision deployment mechanisms on the Moskitt bundles. First, 54 of the Moskitt bundles were considered, representing more than 20000 method declarations. About 2500 FSMs were extracted from these bundles (one per triggering event) in around 6 minutes and 10 seconds. The size of the extracted FSMs vary from 2 states up to 1381 states. However, 3 FSMs were discarded as their non deterministic version has more than 50000 states[3].

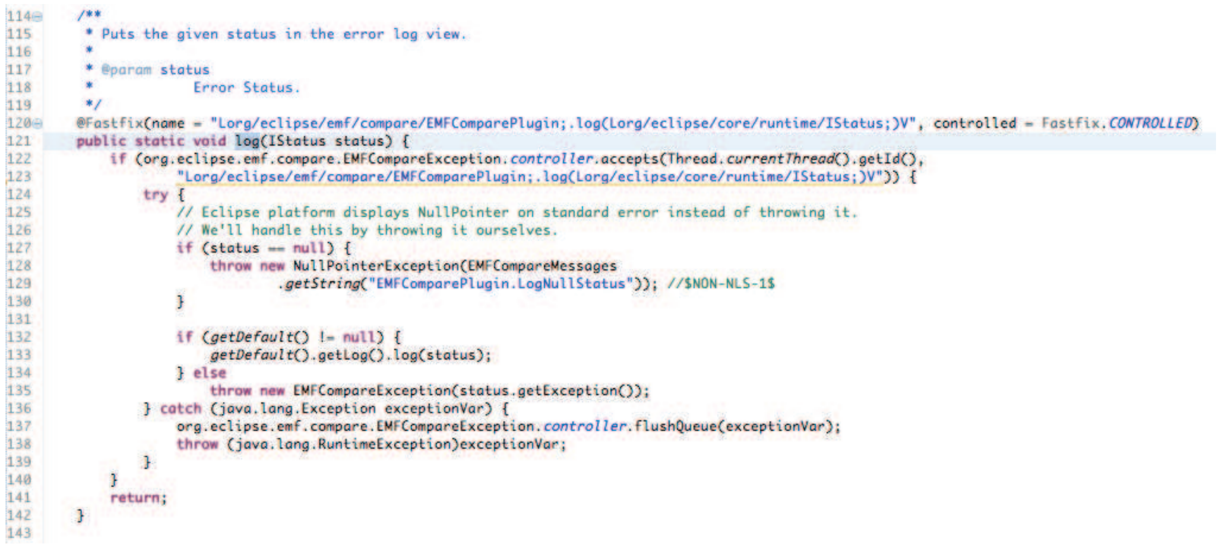
Finally, Figure 4.2 represents the result of the supervision deployment mechanism in the *log* method of the *EMFComparePlugin*. Line 122 and 123 show the call to the *accepts* method from the supervisor. If this method returns *true*, then the method intent to execute the contents of the *try* statement in Line 124. The contents

[3]In this work, a special version of the determinisation algorithm is used which does not ensure an equivalent behaviour to the initial one. However, this algorithm complexity is linear rather than exponential in the classical case.

Nb Bundle	54
Nb Fsm	2492
Nb Methods	21525
Extraction Time	370749 ms
Avg Min FSM Size	2
Avg Max FSM Size	85

Table 4.1: Application of the model extraction and supervision deployment mechanisms to Moskitt. Time is in millisecond and FSM sizes represent numbers of states.

of this *try* clause represent the initial body of the *log* method. If an un-handled exception occurs during the execution of the try clause, then it is caught and the behaviour recorded by the supervisor at runtime and leading to this exception is flushed into a log file for further analysis and patch generation. Moreover, the exception is thrown again in case other deployed supervisors need to be aware of its existence.



```java
/**
 * Puts the given status in the error log view.
 *
 * @param status
 *            Error Status.
 */
@Fastfix(name = "Lorg/eclipse/emf/compare/EMFComparePlugin;.log(Lorg/eclipse/core/runtime/IStatus;)V", controlled = Fastfix.CONTROLLED)
public static void log(IStatus status) {
    if (org.eclipse.emf.compare.EMFCompareException.controller.accepts(Thread.currentThread().getId(),
            "Lorg/eclipse/emf/compare/EMFComparePlugin;.log(Lorg/eclipse/core/runtime/IStatus;)V")) {
        try {
            // Eclipse platform displays NullPointer on standard error instead of throwing it.
            // We'll handle this by throwing it ourselves.
            if (status == null) {
                throw new NullPointerException(EMFCompareMessages
                        .getString("EMFComparePlugin.LogNullStatus")); //$NON-NLS-1$
            }

            if (getDefault() != null) {
                getDefault().getLog().log(status);
            } else
                throw new EMFCompareException(status.getException());
        } catch (java.lang.Exception exceptionVar) {
            org.eclipse.emf.compare.EMFCompareException.controller.flushQueue(exceptionVar);
            throw (java.lang.RuntimeException)exceptionVar;
        }
    }
    return;
}
```

Fig. 4.2: A Moskitt method automatically instrumented in order to enable Supervisory Control.

This example shows the feasibility of applying the pre-deployment phase of our proposed approach on an industrially relevant application. Work such as [24] ensures the feasibility of the supervisory control algorithm on concurrent FSMs such as the ones extracted from Moskitt.

5. Challenges. The control theoretic self-healing approach poses several challenges. Some of them are discussed in this section and related to current research efforts. Most of the challenges under consideration are due to system complexity. Complexity relates to the system size, the system model size, the efficiency of the analyses and supervisor synthesis as well as the need for a low overhead during runtime execution.

The approach in Sect. 3 is flexible enough to allow for complexity reduction by considering only sub-parts of the system to be observed, controlled and modeled and also by approximating the system and control objective models through abstractions. However, reducing the amount of information available to the framework described in Figure 3.2 alters the quality of the supervisors, that can be automatically synthesized and therefore the relevance of the self-healing solution to be applied. Therefore trade-offs between scalability and relevance of the approach have to be determined, posing several challenges. For this purpose challenges related to system observability and controllability, to system modeling, to designing control objective (related to automatic diagnosis), to concurrency and to corrections to be applied (related to automatic repair) are discussed in the rest of this section.

5.1. Finite State Machines and Variables. In Sect. 3 we represent application models as Finite State Machines, where the transitions represent method calls. Although this view of the system behaviours takes into account past executions in order to decide on the control actions to be performed, it does not explicitly take into account system variables. This approach has an interesting upside: the state space of the model is in general smaller than the state space of the application. Without considering system variables, the states of the model do not encode a possible tuple of values of the application variables. Instead states only encode control-flow information (branchings and loops) of the program (as illustrated in Figure 3.3), reducing the model state space.

The downside of this approach is that information on the system behaviours is not as accurate as if variable values were taken into account. For instance, disabling the occurrence of a method call may be dependent on the values of the parameters with which the method is called (if any). Therefore, taking into account some of the application variables into the approach while preserving its scalability is an important but challenging tasks.

Several works have considered supervisory control on FSM with variables: [58, 57, 38, 36, 23]. Although Extended Finite State Machines offer a compact way of representing potentially large, or even infinite system state spaces, the supervisor synthesis takes into consideration the system state space itself. In order to tackle this issue, abstractions of the variable values rather than the possible values themselves should be considered for analysis. This can be done in the same spirit as for Abstract Interpretation ([12]) or data obfuscation techniques (e.g. [4]). Obfuscation techniques aim to abstract the actual variable values into *restricted domains*. Using an FSM makes it easier to calculate the restricted domain of each variable at each point. As transitions that correspond to tests and branches on application variables are performed in the application model, the conjunction of the conditions applied to each variable can be calculated, resulting in the conditions needed to reach the particular point, i.e. the path condition. Naturally, the path condition is a result of the particular values of the program's variables: if the path condition includes the clause $x > 0$ this means that x was tested for being positive somewhere along the execution path and indeed it was positive. Implicitly, the path condition obfuscates the specific variable values for the execution.

5.2. Automatic Recovery. In its basic form, the approach described in Sect. 3 generally requires that the application is restarted in order to take new supervisors into account. This ensures a proper monitoring of the system by the new supervisor. Restarting the application sets the system behaviour model to its initial state. This ensures that the new supervisor can be applied to the system: when it exists a supervisor can always be applied from the system's initial state. One challenge for our approach consists of providing an automated means for avoiding the application relaunch whenever a new supervisor is to be applied. This challenge can be tackled by considering checkpointing techniques such as described in Sect. 2.3.

Checkpointing an entire application is time consuming. In order to lower the rate (and cost) of checkpointing, full checkpoints of the whole application may be complemented with intermediate incremental checkpoints [22] of the memory pages or objects that have changed since the latest full checkpoint. However, the main challenge for checkpointing in a supervised application is to synchronize the application states with model states. Code instrumentation can be used in order to annotate the checkpointing data with the corresponding application model state. In this way both application and model can easily be restarted at the same point. When rollbacks are performed together with a modification of the supervisor (e.g. so that the system does not run towards the previously occurred error), it may not be possible to restart a supervised application at the latest checkpoint. The supervisor model may indeed have been modified so that the model states associated with the latest checkpoint no longer exist. This problem can be sidestepped by rolling the application back to a point where the application execution does not include any state of the supervisor model that has been modified. This can be verified by storing, with each checkpoint, the current supervisor model state as well as all the states that have been visited before. If, when the supervisor model is changed the list of modified states is also stored, then it becomes possible to choose a checkpoint that does not include any modified states.

5.3. Designing Control Objectives. Our proposed approach relies on the synthesis of supervisors from a model of the system behaviours and a control objective. This control objective is represented by a FSM and encodes safety properties over the system behaviours. It is possible for instance to describe what methods must not be executed after some given executions. If the control objective also provides information on the variables of the system, then it allows to describe complex conditions under which some method calls must not be executed.

As mentioned in Sect. 3.1 and illustrated in Figure 3.2, the control objective may be obtained manually and automating its design is a difficult challenge.

Some result in this direction have been obtained in [25] in the specific case of un-handled exceptions. As a general matter, tackling the automatic design of control objective is very much related to automatic fault and anomaly detection (e.g. [10]) as well as automatic diagnosis. Specification mining techniques ([40]) can also be employed in order to extract from the observed undesired trace the pattern that characterize the occurrence of an error.

6. Conclusion. This document deals with software self-healing as investigated in the FastFix FP7 EU project, and focuses on corrective maintenance. A brief state-of-the-art on self-healing is presented and concludes that the research achieved so far is better suited for adaptive and perfective maintenance rather than corrective maintenance.

This work introduces a control theoretic approach which offers a solution to self-healing for corrective maintenance. We describe its different phases: model extraction, supervision deployment and runtime supervision. Results about the feasibility of applying this approach on an industrially relevant system are presented. Finally this paper points out the challenges related to the proposed approach, such as the automatic design of control objective and improving on the application models.

REFERENCES

[1] *Fastfix project consortium: Fastfix project homepage, www.fastfixproject.eu/.*

[2] *The moskitt project: Homepage, http://www.moskitt.org/eng/moskitt0/.*

[3] *Osgi eclipse: Homepage, http://www.eclipse.org/osgi/.*

[4] D. Bakken, R. Rarameswaran, D. Blough, A. Franz, and T. Palmer, *Data obfuscation: anonymity and desensitization of usable data sets*, Security & Privacy, IEEE, 2 (2004), pp. 34–41.

[5] G. Candea and A. Fox, *Crash-only software*, in Proceedings of the 9th Workshop on Hot Topics in Operating Systems (HotOS IX), 2003, pp. 12–20.

[6] A. Carzaniga, A. Gorla, N. Perino, and M. Pezzè, *Automatic workarounds for web applications*, in FSE, 2011.

[7] A. Carzaniga, A. Gorla, and M. Pezzè, *Healing Web applications through automatic workarounds*, International Journal on Software Tools for Technology Transfer (STTT), 10 (2008), pp. 493–502.

[8] ———, *Self-healing by means of automatic workarounds*, in Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems, ACM, 2008, pp. 17–24.

[9] C. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, Kluwer Academic Publishers, 1999.

[10] V. Chandola, A. Banerjee, and V. Kumar, *Anomaly detection: A survey*, ACM Computing Surveys (CSUR), 41 (2009), pp. 1–58.

[11] J. Corbett, M. Dwyer, J. Hatcliff, S. Laubach, C. Pasareanu, and H. Zheng, *Bandera: Extracting finite-state models from Java source code*, in Software Engineering, 2000. Proceedings of the 2000 International Conference on, IEEE, 2002, pp. 439–448.

[12] P. Cousot and R. Cousot, *Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints*, in Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Los Angeles, California, 1977, ACM Press, New York, NY, pp. 238–252.

[13] W. Cui, M. Peinado, H. Wang, and M. Locasto, *Shieldgen: Automatic data patch generation for unknown vulnerabilities with informed probing*, in Security and Privacy, 2007. SP '07. IEEE Symposium on, May 2007, pp. 252 –266.

[14] M. Davidsen and J. Krogstie, *Information systems evolution over the last 15 years*, in Advanced Information Systems Engineering, Springer, 2010, pp. 296–301.

[15] R. de Lemos, *ICSE 2003 WADS Panel: Fault Tolerance and Self-Healing*, (2003).

[16] R. Debouk, S. Lafortune, and D. Teneketzis, *Coordinated decentralized protocols for failure diagnosis of discrete event systems*, Discrete Event Dynamic Systems, 10 (2000), pp. 33–86.

[17] B. Demsky and M. Rinard, *Automatic detection and repair of errors in data structures*, in Proceedings of the 18th annual ACM SIGPLAN conference on Object-oriented programing, systems, languages, and applications, ACM, 2003, pp. 78–95.

[18] ———, *Data structure repair using goal-directed reasoning*, in Proceedings of the 27th international conference on Software engineering, ACM, 2005, pp. 176–185.

[19] M. deQueiroz and J. Cury, *Modular supervisory control of large scale discrete-event systems*, in Discrete Event Systems: Analysis and Control. Proc. WODES'00, Kluwer Academic, 2000, pp. 103–110.

[20] S. Dobson, R. Sterritt, P. Nixon, and M. Hinchey, *Fulfilling the vision of autonomic computing*, Computer, 43 (2010), pp. 35–41.

[21] B. Elkarablieh and S. Khurshid, *Juzi*, in Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on, IEEE, 2009, pp. 855–858.

[22] E. Elnozahy, D. Johnson, and W. Zwaenepoel, *The performance of consistent checkpointing*, in Reliable Distributed Systems, 1992. Proceedings., 11th Symposium on, IEEE, pp. 39–47.

[23] B. Gaudin and P. Deussen, *Supervisory control on concurrent discrete event systems with variables*, American Control Conference, 2007. ACC'07, (2007), pp. 4274–4279.

[24] B. Gaudin and H. Merchand, *An efficient modular method for the control of concurrent discrete event systems: A language-based approach*, Discrete Event Dyn Syst, 17 (2007), pp. 179–209.

[25] B. Gaudin, E. Vassev, M. Hinchey, and P. Nixon, *A control theory based approach for self-healing of un-handled runtime exceptions*, in 8th International Conference on Autonomic Computing (ICAC 2011), Karlsruhe, Germany, 06/2011 2011.

[26] D. Ghosh, R. Sharman, H. Raghav Rao, and S. Upadhyaya, *Self-healing systems - survey and synthesis*, Decis. Support Syst., 42 (2007), pp. 2164–2185.

[27] N. Gruska, A. Wasylkowski, and A. Zeller, *Learning from 6,000 projects: lightweight cross-project anomaly detection*, in ISSTA '10: Proceedings of the 19th international symposium on Software testing and analysis, New York, NY, USA, 2010, ACM, pp. 119–130.

[28] J. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury, *Feedback control of computing systems*, Wiley-IEEE Press, 2004.

[29] C. Hood and C. Ji, *Proactive network-fault detection [telecommunications]*, Reliability, IEEE Transactions on, 46 (2002), pp. 333–341.

[30] D. Jeffrey, M. Feng, N. Gupta, and R. Gupta, *Bugfix: A learning-based tool to assist developers in fixing bugs*, in Program Comprehension, 2009. ICPC'09. IEEE 17th International Conference on, IEEE, 2009, pp. 70–79.

[31] M. Jiang, M. Munawar, T. Reidemeister, and P. Ward, *Automatic fault detection and diagnosis in complex software systems by information-theoretic monitoring*, in Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP International Conference on, IEEE, 2009, pp. 285–294.

[32] J. O. Kephart and D. M. Chess, *The vision of autonomic computing*, Computer, 36 (2003), pp. 41–50.

[33] J. O. Kephart and D. M. Chess, *The vision of autonomic computing*, IEEE Computer, 36 (2003), pp. 41–50.

[34] A. D. Keromytis, *Characterizing self-healing software systems*, in In Proceedings of the 4th International Conference on Mathematical Methods, Models and Architectures for Computer Networks Security (MMM-ACNS, 2007.

[35] N. Kolettis and N. D. Fulton, *Software rejuvenation: Analysis, module and applications*, in Proceedings of the 25th International Symposium on Fault-Tolerant Computing (FTCS-25), 1995, pp. 381–395.

[36] R. Kumar and V. Garg, *On computation of state avoidance control for infinite state systems in assignment program framework*, Automation Science and Engineering, IEEE Transactions on, 2 (2005), pp. 87–91.

[37] S. Laster and A. Olatunji, *Autonomic Computing: Towards a Self-Healing System*, (2007).

[38] T. Le Gall, B. Jeannet, and H. Marchand, *Supervisory control of infinite symbolic systems using abstract interpretation*, in 44nd IEEE Conference on Decision and Control (CDC'05) and Control and European Control Conference ECC 2005, Seville (Spain), December 2005, pp. 31–35.

[39] B. P. Lientz, E. B. Swanson, and G. E. Tompkins, *Characteristics of application software maintenance*, Commun. ACM, 21 (1978), pp. 466–471.

[40] D. Lo, S. Khoo, and C. Liu, *Mining temporal rules for software maintenance*, Journal of Software Maintenance and Evolution: Research and Practice, 20 (2008), pp. 227–247.

[41] M. Locasto, K. Wang, A. Keromytis, and S. Stolfo, *Flips: Hybrid adaptive intrusion prevention*, in Proceedings of the 8th International Symposium on Recent Advances in Intrusion Detection (RAID 2005), 2005, pp. 82–101.

[42] M. Malik, K. Ghori, B. Elkarablieh, and S. Khurshid, *A case for automated debugging using data structure repair*, in Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering, IEEE Computer Society, 2009, pp. 620–624.

[43] G. Novark, E. Berger, and B. Zorn, *Exterminator: Automatically correcting memory errors with high probability*, in Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation, ACM, 2007, pp. 1–11.

[44] J. Perkins, S. Kim, S. Larsen, S. Amarasinghe, J. Bachrach, M. Carbin, C. Pacheco, F. Sherwood, S. Sidiroglou, G. Sullivan, et al., *Automatically patching errors in deployed software*, in Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, ACM, 2009, pp. 87–102.

[45] H. Psaier and S. Dustdar, *A survey on self-healing systems: approaches and systems*, Computing, 91 (2011), pp. 43–73. 10.1007/s00607-010-0107-y.

[46] F. Qin, J. Tucek, Y. Zhou, and J. Sundaresan, *Rx: Treating bugs as allergies—a safe method to survive software failures*, ACM Transactions on Computer Systems (TOCS), 25 (2007), p. 7.

[47] J. Radatz, *IEEE standard glossary of software engineering terminology*, IEEE Std 610121990, 121990 (1990).

[48] P. J. Ramadge and W. Wonham, *Supervision of discrete event processes*, in Proc. of 21st IEEE Conf. Decision and Control, Orlando, FL, Dec. 1982, pp. 1228–1229.

[49] ———, *Supervisory control of discrete event processes*, in Feedback Control of Linear and Nonlinear Systems, vol. 39 of LNCIS, Springer-Verlag , Berlin, Germany, 1982, pp. 202–214.

[50] O. Raz, P. Koopman, and M. Shaw, *Enabling automatic adaptation in systems with under-specified elements*, in WOSS '02: Proceedings of the first workshop on Self-healing systems, New York, NY, USA, 2002, ACM, pp. 55–60.

[51] R. Reiter, *A theory of diagnosis from first principles*, Artificial Intelligence, 32 (1987), pp. 57–95.

[52] G. D. Rodosek, K. Geihs, H. Schmeck, and B. Stiller, *Self-healing systems: Foundations and challenges*.

[53] M. Salehie and L. Tahvildari, *Self-adaptive software: Landscape and research challenges*, Transactions on Autonomous and Adaptive Systems (TAAS, 4 (2009).

[54] A. Sengupta and A. Dahbura, *On self-diagnosable multiprocessor systems: diagnosis by the comparison approach*, Computers, IEEE Transactions on, 41 (2002), pp. 1386–1396.

[55] O. Shehory, *A self-healing approach to designing and deploying complex, distributed and concurrent software systems*, in ProMAS'06: Proceedings of the 4th international conference on Programming multi-agent systems, Berlin, Heidelberg, 2007, Springer-Verlag, pp. 3–13.

[56] S. Sidiroglou, O. Laadan, C. Perez, N. Viennot, J. Nieh, and A. D. Keromytis, *Assure: automatic software self-healing using rescue points*, in ASPLOS '09: Proceeding of the 14th international conference on Architectural support for programming languages and operating systems, New York, NY, USA, 2009, ACM, pp. 37–48.

[57] M. Skoldstam, K. Akesson, and M. Fabian, *Modeling of discrete event systems using finite automata with variables*, in Decision and Control, 2007 46th IEEE Conference on, IEEE, 2007, pp. 3387–3392.

[58] ———, *Supervisory control applied to automata extended with variables-revised*, Relatório técnico, Goteborg: Chalmers Uni-

versity of Technology, (2008).

[59] A. Smirnov and T.-c. Chiueh, *Automatic patch generation for buffer overflow attacks*, in IAS '07: Proceedings of the Third International Symposium on Information Assurance and Security, Washington, DC, USA, 2007, IEEE Computer Society, pp. 165–170.

[60] M. Sullivan and R. Chillarege, *Software defects and their impact on system availability-a study of field failures in operating systems*, in Proceedings of the 21st International Symposium on Fault-Tolerant Computing (FTCS-21), 1991, pp. 2–9.

[61] Y. Wei, Y. Pei, C. A. Furia, L. S. Silva, S. Buchholz, B. Meyer, and A. Zeller, *Automated fixing of programs with contracts*, in ISSTA '10: Proceedings of the 19th international symposium on Software testing and analysis, New York, NY, USA, 2010, ACM, pp. 61–72.

[62] Y. Willner and M. Heymann, *Supervisory control of concurrent discrete-event systems*, International Journal of Control, 54 (1991), pp. 1143–1169.

[63] W. M. Wonham, *Notes on control of discrete-event systems*, Tech. Report ECE 1636F/1637S, Department of Electrical and Computer EngineeringUnivertsity of Toronto, July 2003.

Edited by: Dana Petcu and Daniela Zaharie
Received: March 1, 2012
Accepted: April 15, 2012

Scalable Computing: Practice and Experience
Volume 13, Number 1, pp. 21–28. http://www.scpe.org

ISSN 1895-1767
© 2012 SCPE

1

SIMULATION OF COMMUNICATION AND COOPERATION IN MULTISPECIES BACTERIAL COMMUNITIES WITH AN AGENT BASED MODEL

DÓRA BIHARY[1*], ÁDÁM KERÉNYI[2*], ZSOLT GELENCSÉR[1], SERGIU NETOTEA[3], ATTILA KERTÉSZ-FARKAS[4], VITTORIO VENTURI[4] AND SÁNDOR PONGOR[1,2,4]

Abstract.
Members of bacterial communities communicate and cooperate via diffusible chemical materials they emit into the environment, and at the same time, they also compete for nutrients and space. Agent-based models (ABMs) are useful tools for simulating the growth of communities containing multiple interacting microbial species. In this work we present numerical indices characterizing spatial distribution and the fitness of competing bacterial species in an ABM and we present data on how these indices can be used to visually summarize large scale simulation experiments. Preliminary results show bacterial agents utilizing different nutrients but sharing communication signals and public goods can form stable mixed communities in which the species grow faster than any of the single species alone.

Key words: quorum sensing, *Pseudomonas aeruginosa*, hybrid model, statistics, segregation, fitness

1. Introduction, state-of-the-art.

Multispecies microbial communities are now recognized as a major form of bacterial life. These communities (such as the gastrointestinal flora, the microflora of dental cavities, the rhizosphere around plant roots or the large microbial mats on the seafloor) contain more than one species. Computer simulations play an important role in the study of these communities since it is extremely complicated to collect reliable data on the size and growth dynamics of free-living bacterial communities.

The interaction between individual bacteria in a community is often based on the exchange of diffusible signals, the best known example of which is a mechanism called quorum sensing (QS) [1, 2]. In this mechanism, signaling materials secreted by the bacteria are supposed to spread in the environment by diffusion. The concentration of signals regulates the behavior of bacteria, which results in collective patterns of behavior, such as coordinated movement (e.g. swarming), secretion of specific materials, resistance to antibiotics, etc. When the concentration of the secreted signal is greater than a certain threshold, bacteria, such as *Pseudomonas aeruginosa*, switch from low to high metabolic activity, they increase the amount of secreted signaling molecules and they also start to secret other molecules, frequently referred to as "public goods" or simply "factors" (e.g. surfactants, enzymes, siderophores), which facilitates movement and nutrient uptake [2, 3]. As a result, the colony changes behavior, for instance it starts to grow and expand. In some cases, this is accompanied by a swarming motion of the cells.

There are various approaches for modeling the growth of bacterial communities. Continuous models represent both the nutrients and the bacterial colony as continuous quantities described by reaction-diffusion equations [3]. Agent-based models consider bacteria as individuals capable of nutrient uptake, movement and cell-division [4, 5]. In agent based models, the nutrients are often considered as diffusing materials described by reaction-diffusion equations - these models are specifically called "hybrid models" as they combine individual-based agents with diffusing materials. Continuous and agent-based hybrid models were used primarily to show that colony shapes (especially the well known fractal-like or circular shapes) can be reproduced by simple models [3, 4, 5].

Recently we adapted the hybrid methodology for describing the behavior of QS bacteria [6, 7]. Briefly, agents representing individual bacteria move randomly on a 2D plane in this model, and they secrete two kinds of materials, a signal S and a factor F (public goods material) that both spread via diffusion on the 2D plane. Agents have different "physiological" states and they switch between states depending on the local concentration of S and F. In the ground state (S and F below threshold), nutrient uptake, movement and signal production is at a low level. In the activated state (S above threshold, F below threshold), the production of public goods (F) starts, and signal production is upgraded to a higher level. In the quorum state, signal production is high, production of public goods is high, and movement and nutrient uptake are also upgraded to a higher level. In

[1]Faculty of Information Technology, Pázmány Péter Catholic University, Práter u. 50/a. 1083 Budapest, Hungary, [2]Biological Research Centre of the Hungarian Academy of Sciences, H-6726, Szeged, Temesvári krt. 62, Hungary, [3]Umeå University, SE-901 87 Umeå, Sweden, [4]International Centre for Genetic Engineering and Biotechnology, Padriciano 99, I-34012 Trieste, Italy, pongor@icgeb.org, *These authors contributed equally.

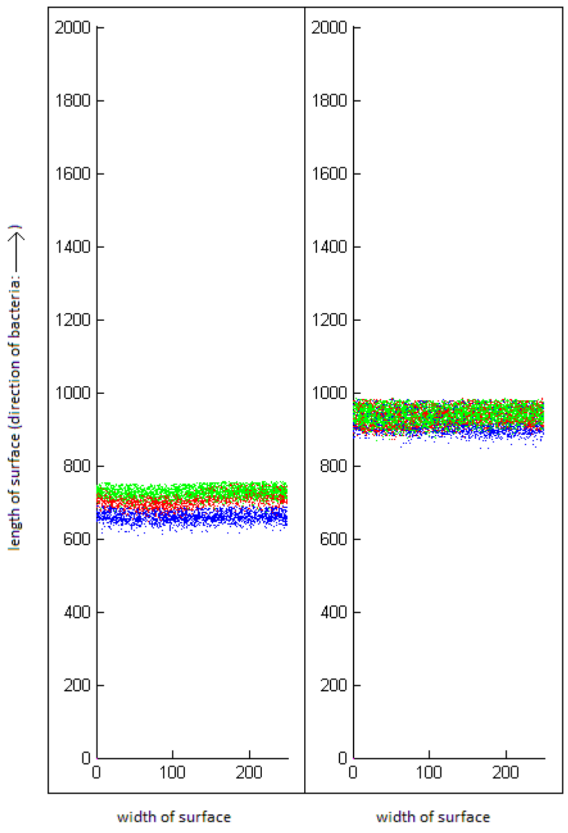


Fig. 1.1: Simulation of segregation (left) and co-localization (right) of mixed, swarming communities. In these experiments, the three species consumed nutrients that were not accessible for the other species. In the left lane, the species could neither communicate nor cooperate with the other species. In the right lane, the species could communicate and cooperate with each other.

summary, the cells' behavior (nutrient intake, movement, production of signals and public goods) alternates between two levels, according to certain threshold criteria. In a simulation run, a small number (usually 2000) of ground state agents are placed randomly at the bottom of a cylindrical surface (Figure 1) and then let to perform nutrient uptake, movement etc. according to their individual programs. When the cells accumulate a certain quantity of nutrients, they divide and the progeny increases the number of living cells. Cells that run out of nutrients "die". This is a highly simplified model in which all parameters are in arbitrary units, and time is represented as discrete time steps. Nevertheless the model is able to reproduce seemingly complex behavior patterns that occur in natural bacterial communities: a) Switching from ground state to an active (quorum) state is dependent on cell density - this is the fundamental hypothesis of quorum sensing. b) The colony is capable to track external signals - this phenomenon occurs in nature when plant roots recruit bacteria from the surrounding rhizosphere. c) Bacteria that respond to signal but are unable to produce it, are capable to form viable communities with healthy (wild type) cells that both produce and sense the signal. d) Mutants that do not cooperate (do not produce public goods) will collapse a community of healthy (wild type) cells [6, 7].

The ultimate goal of our modeling project is to get insights into the stability criteria that keep multispecies communities stable in time. Previously we showed that synthetic bacterial communities (e.g. those formed by cells of two, selected species) can "combine the skills of the participants", i.e. a mixed community will be able to withstand to conditions in which their constituent species cannot survive alone [7]. Recently we also found that bacterial species that share signals can form stable communities in nature [8]. Our hypothesis is that the extent to which two species can share signals, public goods, nutrients are a crucial factor in the stability of a community, and coexisting communities will either co-localize (form truly mixed communities) or will segregate within the available space. In order to analyze these phenomena, we need quantitative descriptors to characterize the mixed agent communities in biological terms, which is the subject of this work. The goal of the present work is not necessarily to develop novel indices for all of the interesting quantities, rather we seek to test the applicability of the indices. One of our particular goals is to develop suitable visualization techniques that can summarize the behaviour of an agent community throughout the parametric space. In Section 2 we introduce the biological terminology used in this paper. In Section 3 we describe an index developed for the segregation of agent communities. Section 4 describes the concept of relative fitness applied to our agent-based system. In Section 5 we introduce indices that can be used to characterize the correlated motion within communities. In Section 6 and 7 we apply these indices to the modeling of a mixed agent community in which the members are mutually dependent on each other. Finally, Section 8 contains the conclusions.

2. Biological terminology. *Quorum sensing (QS)* is the comprehensive name of the mechanism by which bacteria sense the presence of other members of their species. The best studied version of this phenomenon is based on the secretion of diffusible signals *(QS signals)* and other materials called public goods. Public goods are materials that are useful not only for the cell that secretes them but for any cell that can use them. This is often referred to as *cooperation*, since a cell cooperates with the community by producing materials. In contrast, signaling via diffusible signals is often referred to as *communication*. Cells that are able to both communicate (i.e. produce and sense signals) and cooperate (produce and sense public goods) are referred to as wild type or wt cells. Cells that are different from wt cells are briefly referred to as mutants since it is normally assumed that they lost some of the abilities (or gained new ones) by mutation. QS is studied perhaps in the greatest detail in the bacterium *Pseudomonas aeruginosa* which is an ubiquitous, opportunistic pathogen causing potentially lethal infections to humans. Other members of the *Pseudomonas* genus are ubiquitous in soil and water and are sometimes beneficial to host organisms such as plants.

3. Spatial segregation of agent communities. In the simulation model the space is represented as a cylindrical surface on which the bacterial community starts from the bottom and proceeds upwards while consuming the nutrients found on the surface. We can follow the collapse or survival of a species by counting the number of cells at each step. On the other hand, the spatial distribution of cells also changes in time: some communities segregate while they move (Figure 1.1, left) while others remain co-localized (right). Mitri et al. [9] have described this behavior by an intuitive segregation index based on the work of Nadell et al. [10] which depends on counting an arbitrary number of nearest neighbors for each agent. This is an $O(n^2)$ algorithm with respect to the number n of agents, and since the calculation has to be repeated at every time step for communities as large as say 50 thousand members, we were looking for alternative ways to describe spatial segregation. In order to develop a segregation index that can be calculated in a more time efficient manner, we take advantage of the fact that space in our simulation is divided into squares that form a matrix-like lattice. In each square we can count the number of bacteria from each species. E.g. for three species (denoted i, ii and iii, respectively) we can calculate $n_1(i)$, $n_2(i)$, $n_3(i)$ in the i^{th} square, so we can express the fraction (or percentage) of each species within the square. If a population is segregated, this fraction is almost 1 for one of the species and almost 0 for the two other species, so we define the segregation index as the maximum fraction of a species - in other words, the fraction of the dominant species - within a given spatial unit (in the 2D plane). We get a more representative value if we weight these fractions with the total number of bacteria in the actual unit of space. By this step we get the segregation coefficient which is the average of the number of the dominant species in the given spatial units, divided by the number of the total population.

$$SG = \frac{\sum_i \max(n_i)}{N_{population}}, \tag{3.1}$$

where $\max(n_i)$ is the cell number of the dominant species within the i^{th} space unit, the denominator it the total number of the population (including all species). For a randomly mixed community (such as shown in Figure 1.1, right), this quantity will approach the reciprocal of the number of species present which allows us to construct a [0,1] numerical index as follows:

$$SGN = \frac{\left(SG - \frac{1}{N_{species}}\right)}{\left(1 - \frac{1}{N_{species}}\right)}, \tag{3.2}$$

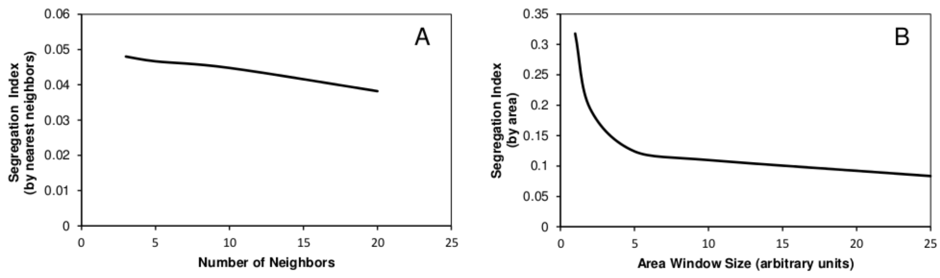


Fig. 3.1: Calculation of the segregation index for a homogeneous (non-segregated community). The Y axis is the normalized segregation index. A. Normalized segregation index calculated by nearest neighbors vs. the number of neighbors included in the calculation. B. Normalized segregation index calculated surface area vs. the size of the surface area included in the calculation.

where SGN is the normalized segregation index calculated at a certain time step and $N_{species}$ is the number of agent species present. This quantity is between zero and 1.0. $SG = 1$ if the population is segregated, in a fashion seen in Figure 1.1, left. $SG = 0$ means that the populations are co-localized, in a fashion seen in Figure 1.1, right. The calculation of this index is not time consuming, it has a time complexity of $O(n)$. Whether we calculate the segregation index using the number of nearest neighbors, or based on surface area units, we have an arbitrary parameter in the calculation - the number of neighbors in the first case, and in the second case, the size of the surface included in the calculation. We tested the behavior of the indices on binary communities segregated to various extents. On well segregated communities, both calculations gave values of 1.0 throughout the entire parameter range (3 to 20 nearest neighbors or 1 to 25 units of surface area, data not shown). On homogeneous communities the values depended on the parameters (Figure 3.1). As expected, the calculation by surface area was about 2 orders of magnitudes faster than the one calculated by nearest neighbors. The area-dependent calculation is sensitive to the size of the area used for the calculation, while the nearest neighbor-dependent index is dependent to the number of neighbors included in the calculation.

The dependence on the window size was practically the same for various window shapes, we found no difference between 50x1 square, 25x2 square, or 10x5 square windows. Nevertheless, the values are parameter dependent, as shown in Figure 3.1., so it is recommended to use SGN on a comparative bases, *i.e.* for populations of the same size and density. In practice we calculated SG indices for the horizontal rows (50 area units) of the 2D space matrix that maps the space in Figure 1.1. SGN shows typical saturation kinetics as a function of time. If the populations segregate, SGN converges to a value above 0.8. If the populations co-localize, the value remains low, an example is shown in Figure 4.1.

4. Relative fitness calculation. In biology, the fitness of a population (say a colony of bacteria) is calculated from the growth of the population achieved in a given time (for a recent review see [11]). In fact, since the work of Darwin, many increasingly sophisticated methods were developed to define fitness.

From these, we chose a simple formula that has been used for agent communities by Mitri et al. [9]:

$$F = \frac{1}{\Delta t} \log_2 \frac{N_{end}}{N_{start}}, \tag{4.1}$$

where F is the fitness, N_{start}, and N_{end}, are the size of the population at the start and end of the experiment respectively, Δt is the elapsed time. For increasing population the logarithm is positive, however for decreasing populations it becomes negative. Fitness is a dimensionless number which is often represented on a relative scale, in comparison with the fitness of a reference species. This is especially handy in our case since we can easily select one of the species, e.g. the wild type (wt) species as the reference and then we get:

$$F_{rel} = \frac{\log_2 \left(\frac{N_{end}}{N_{start}} \right)}{\log_2 \left(\frac{N_{end,wt}}{N_{start,wt}} \right)}, \tag{4.2}$$

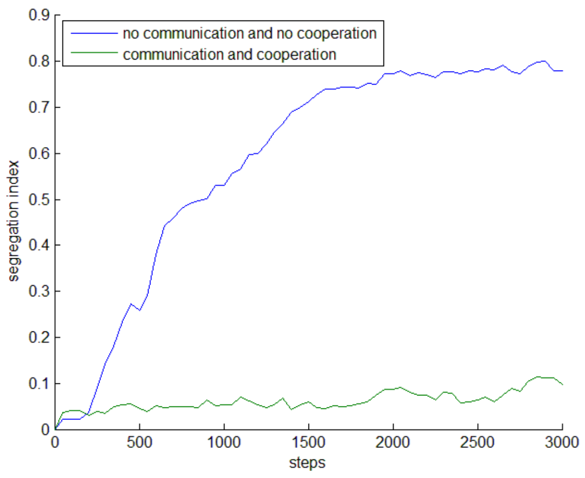


Fig. 4.1: Example of segregating (blue) and co-localizing (green) communities. The Y axis is the normalized segregation index calculated by surface area (10 units), the X axis shows time steps within the simulation. The experiment is the same as described in the legend of Figure 1.1

where F_{rel} is the relative fitness, $N_{end,wt}$ and $N_{start,wt}$ are the reference values for wild type population. The Δt of eqn. 4.1 terms are cancelled by the division. It is important to note that relative fitness is often calculated between two competing species. We followed a different strategy, we used the wild type species growing alone, as the reference. So in this case, the relative fitness will tell us if members of a community grow better or worse if they grow together, in other words, if it is an advantage to be in a community. Table 4.1 shows typical simulation results for a ternary community in which 3 species were put together in equal amounts. The experiment is the same as described in Figure 1.1, i.e. in one case the 3 species can neither communicate, nor cooperate, in the other case they both communicate and cooperate.

The experiments in Table 4.1 show that the relative fitness of all participant species increases if they can communicate (signal sharing) and cooperate (sharing of public goods or factors). In these experiments, all species had their own nutrients that could not be consumed by the other species, so the conclusions may not hold for other conditions.

5. Correlation measures. Ever since the simulation of multi-particle systems became feasible, correlation measures were used in many areas of physics and chemistry. The correlation functions developed in molecular dynamics [12] can be especially easily extended to cellular or animal systems [13, 14]. In this paper, the movement of two agents i, j is called correlated if their velocities point to the same direction. In this case the inner product of the unit vectors of velocity $\vec{v_i}\vec{v_j}$ will be 1.0. For a population of agents we can calculate an aggregate measure by determining the inner product of all i,j, $(j > i)$ agent pairs. The average correlatedness of the velocities of n agents can be expressed as:

Table 4.1: Comparison of relative fitness in communicating, cooperating populations and non-communicating, non-cooperating communities.

	Species 1	Species 2	Species 3
Single wild type population alone	1.000	-	-
Co-localized community (cooperation + communication)	1.2402	1.2304	1.3063
Segregated community (no cooperation, no communication)	1.0062	1.0188	1.0094

Relative fitness F_{rel}.

$$c = \frac{2}{n(n-1)} \sum_{i} \sum_{j>i} \vec{v_i}\vec{v_j}. \tag{5.1}$$

The value of c is close to one for agents moving in a perfectly correlated manner. Randomly moving agent populations have c values close to zero. Adding an increasing amount of noise to the direction of velocity vectors in a perfectly coordinated community will decrease the value of c from 1.00 to around 0.00. In other words, c can be used to characterize the level of order within an agent community. In practice, it is convenient to calculate this measure for a given subset of the agents, e.g. limiting the second summation either to agents within a certain distance boundary, or only to a certain number of neighbors. In statistical physics it is customary to limit the calculations for a distance interval $[r, r+\delta]$ around each agent. Plotting the resulting correlatedness values as a function of r will give an impression on how the movement of agents is correlated with more and more distant neighbors.

If the population of agents consists of two subpopulations, A and B, one can calculate correlation measures following (5.1), either i) for the entire population, or ii) for either of the two populations. In addition, one can calculate the correlatednes iii) between the two populations as follows:

$$c_{AB} = \frac{2}{n(A)n(B)} \sum_{i\in A} \sum_{j\in B} \vec{v_i}\vec{v_j}, \tag{5.2}$$

where $n(A)$ and $n(B)$ denote the number of agents in species A and B, respectively.

A further type of measure, autocorrelation of the velocities characterizes the change of the velocities in time. In analogy to (5.1), the autocorrelation can be calculated as

$$c(\Delta t) = \frac{1}{n} \sum_{i} \vec{v}_{i,t} \vec{v}_{i,t+\Delta t}, \tag{5.3}$$

where Δt is a time interval for which the velocities are compared. Note that eqn. 5.3 refers to the same agent. Formulas (5.1-5.3) provide a variety of interesting visualization possibilities, for instance one can plot various distributions for the population, for the changes in time, etc. In addition, modifying (5.3) so as to calculate the correlation between different individuals provides a measure how one individual follows its neighbors. Plotting this value for the entire populations gives a possibility to pinpoint individuals that are "leaders" followed by their neighbors [15].

6. Case study: Mutually Dependent Species. An interesting phenomenon in the bacterial world is the existence of mutually dependent species, i.e. species that depend on each other. In the agent based model of quorum sensing, this situation can be pictured as species responding to signal and/or public goods of another species. For instance we can define a sharing coefficient [0,1] that determines the sensitivity of a species towards the signal (or public goods) of another species. If this sharing coefficient is zero, the species respond only to their own signal (and public goods). If the sharing coefficient is 1.00, the species respond only to the signal (and public goods) of the other species in the same way as they respond to their own signal (and public goods). The value of 1.00 thus denotes a situation of mutual communication in terms of signal (or mutual cooperation in terms of public goods).

We carried out simulations with two competing species, by systematically varying the values of signal sharing and factor sharing in the entire range. The simulation experiments were allowed to proceed for 5000 steps, and the values of relative fitness, segregation coefficient we calculated as the average for the last 500 steps. The results in Figure 6.1 indicate that the two populations do not segregate at any point of the parameter space, but only fully communicating and cooperating species provide large populations and fitness values approaching or exceeding the value of the reference species (the wild type species which is viable in itself, i.e. it is self-sufficient in terms of signals and public goods). In other words, the simulations confirm the biological intuition that predicts that species completely dependent on another species may not be viable in themselves. On the other hand the results suggest that - in harmony with the results of the previous sections - members of a fully communicating and cooperating interspecies community can be fitter than any of its constituent species growing alone.

7. Case Study: Segregating Species. Some microbial species spontaneously form segregating populations. The segregation can be studied with agents endowed with self-recognition capabilities that can be simply modeled by Lennard-Jones-like (LJ) potentials. Briefly, agents recognizing each other will have a preferred distance resulting from attraction or repulsion calculated from a LJ-like potential, while agents ignoring each other will only repulse each other by forces calculated from a Weeks-Chandler-Andersen-like (WCA) potential. Such binary populations are capable of segregation, and as it is shown in Figure 7.1. In this experiment, the segregation is accompanied by a correlated movement. After the segregation, the movement is uncorrelated which is shown by the correlation coefficient falling to a value very close to zero. This behaviour is highly reminiscent of phase transition phenomena known in statistical physics.

8. Conclusions. In this paper we presented numerical indices for characterizing the relative fitness and spatial segregation-co-localization properties of agent populations forming multi-species consortia. The agent models used in this work are meant to simulate the growth of quorum sensing bacterial species that are known to form multispecies communities. We showed that the indices can be used to visualize the behavior of such complex communities in terms of simple diagrams such as the heat maps shown in Figure 6.1. The preliminary results presented here suggest that communication and cooperation between species feeding on different nutrients result in co-localizing communities in which the participating species are fitter then when living alone.

Correlated movement of animal species has been investigated in a number of fields. Our results show that self-recognition capabilities may be sufficient for inducing segregation of agent populations. These results are preliminary as the calculations were meant to illustrate the use of the numerical indices and visualization principles described here. More detailed analysis will be carried out in the future to confirm the validity and the scope of these findings.

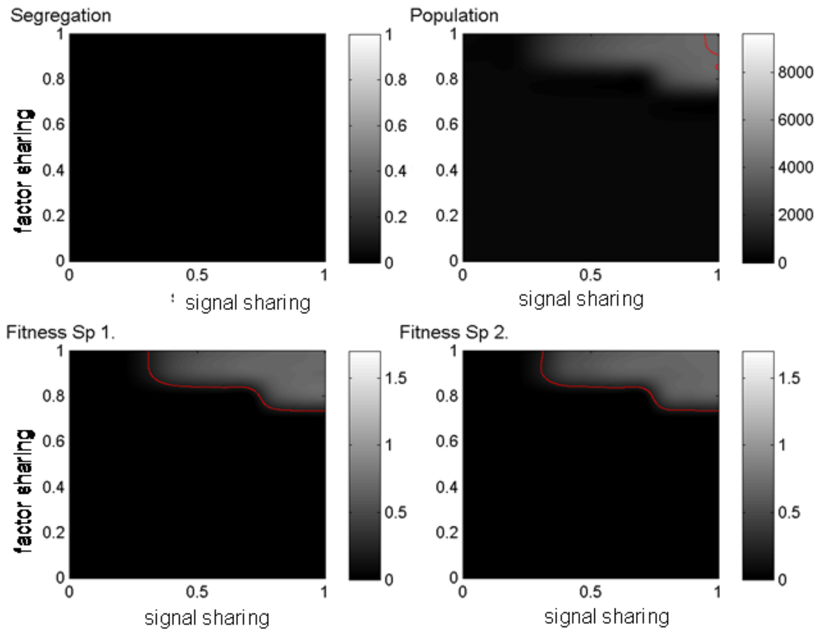


Fig. 6.1: Behavior of mutually dependent species in a binary community. Zero signal sharing (factor sharing) means a species is fully independent from the signal (public goods) of the other species (bottom left corner). Full sharing (a value of 1.0) means that the two species understand the signal and the factor of the other species (top right corner). The figures represent heat maps of (clockwise from top left) segregation coefficient, population size, relative fitness of species 2 and species 1. It is apparent that only mutually cooperating and communicating species are viable in this system (light grey areas at the top left corners of the parameter space). Relative fitness in the corresponding ranges slightly exceeds the *normal values*, i.e. the value of the wild type cells.

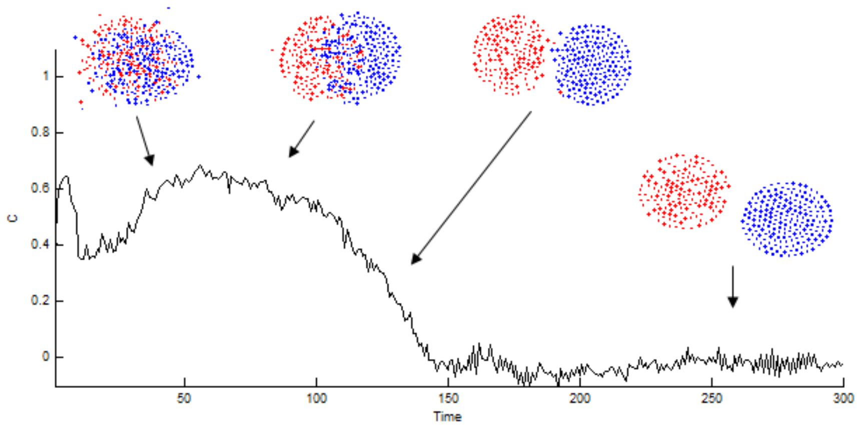


Fig. 7.1: Emergence of correlated movement in spontaneously segregating agent populations. Two randomly moving agent populations, 300 agents each, were put into a random arrangement (inset, left.), c = 0 should be curly equal sign). With time the population starts to segregate and nonzero c values emerge. At the end, the populations are separated and the c value returns to zero.

9. Acknowledgement. Work at the Szeged Biological Center was partially supported by OTKA grant K. 84335 and TÁMOP-4.2.2-08/1-2008-0008. Dóra Bihary and Zsolt Gelencsér are PhD students at Faculty of Information Technology, Pázmány Péter Catholic University, Budapest.

REFERENCES

[1] M. B. Miller and B. L. Bassler. Quorum sensing in bacteria. *Annu Rev Microbiol*, 55:165–99, 2001.
[2] V. Venturi and S. Subramoni. Future research trends in the major chemical language of bacteria. *HFSP J*, 3(2):105–16, 2009.
[3] K. Kawasaki, A. Mochizuki, M. Matsushita, T. Umeda, and N. Shigesada. Modeling spatio-temporal patterns generated by bacillus subtilis. *J Theor Biol*, 188(2):177–85, 1997.
[4] E. Ben-Jacob, I. Cohen, O. Shochet, I. Aranson, H. Levine, and L. Tsimring. Complex bacterial patterns. *Nature*, 373(6515):566–7, 1995.
[5] I. Golding, I Cohen, and E. Ben-Jacob. Spatio-selection in expanding bacterial colonies. *Physica A*, 1999.
[6] S. Netotea, I. Bertani, L. Steindler, A. Kerenyi, V. Venturi, and S. Pongor. A simple model for the early events of quorum sensing in pseudomonas aeruginosa: modeling bacterial swarming as the movement of an "activation zone". *Biol Direct*, 4:6, 2009.
[7] V. Venturi, I. Bertani, A. Kerenyi, S. Netotea, and S. Pongor. Co-swarming and local collapse: quorum sensing conveys resilience to bacterial communities by localizing cheater mutants in pseudomonas aeruginosa. *PLoS One*, 5(4):e9998, 2010.
[8] T. Hosni, C. Moretti, G. Devescovi, Z. R. Suarez-Moreno, M. B. Fatmi, C. Guarnaccia, S. Pongor, A. Onofri, R. Buonaurio, and V. Venturi. Sharing of quorum-sensing signals and role of interspecies communities in a bacterial plant disease. *ISME J*, 5(12):1857–70, 2011.
[9] S. Mitri, J. B. Xavier, and K. R. Foster. Social evolution in multispecies biofilms. *Proc Natl Acad Sci U S A*, 108 Suppl 2:10839–46, 2011.
[10] C. D. Nadell, K. R. Foster, and J. B. Xavier. Emergence of spatial structure in cell groups and the evolution of cooperation. *PLoS Comput Biol*, 6(3):e1000716, 2010.
[11] H. A. Orr. Absolute fitness, relative fitness, and utility. *Evolution*, 61(12):2997–3000, 2007.
[12] B. J. Alder and T. E. Wainwright. Phase Transition for a Hard Sphere System. *The Journal of Chemical Physics*, 27(5):1208–1209, 1957.
[13] Andrea Cavagna, Alessio Cimarelli, Irene Giardina, Giorgio Parisi, Raffaele Santagati, Fabio Stefanini, and Massimiliano Viale. Scale-free correlations in starling flocks. *Proceedings of the National Academy of Sciences of the United States of America*, 107(26):11865–70, 2010.
[14] Tamás Vicsek and Anna Zafeiris. Collective motion, arxiv:1010.5017. 2010.
[15] M. Nagy, Z. Akos, D. Biro, and T. Vicsek. Hierarchical group dynamics in pigeon flocks. *Nature*, 464(7290):890–3, 2010.

Edited by: Dana Petcu and Daniela Zaharie
Received: March 1, 2012
Accepted: April 15, 2012

Scalable Computing: Practice and Experience
Volume 13, Number 1, pp. 29–44. http://www.scpe.org
ISSN 1895-1767
© 2012 SCPE

ENABLING MODEL DRIVEN ENGINEERING OF CLOUD SERVICES BY USING MOSAIC ONTOLOGY

FRANCESCO MOSCATO* AND B. DI MARTINO AND R. AVERSA†

Abstract.
The easiness of managing and configuring resources and the low cost needed for setup and maintaining Cloud services have made Cloud Computing widespread. Several commercial vendors now offer solutions based on Cloud architectures. More and more providers offer new different services every month, following their customers needs. A way to provide a common access to Cloud services and to discover and use required services in Cloud federations is appealing. mOSAIC project addresses these problems by defining a common ontology and it aims at developing an open-source platform that enables applications to negotiate Cloud services as requested by users. Anyway the increasing complexity of services required by users in Cloud Environments usually needs the definition of composite, value added services (VAS). Usage patterns and Use Cases definitions help in defining VAS, but a way to assure that new services reach the required goals with proper qualitative and quantitative properties has to be provided in order to validate design and implementation of composite services. In this paper mOSAIC Ontology is described and the MetaMORP(h)OSY methodology and framework are introduced. The methodology uses Model Driven Engineering and Model Transformation techniques to analyse services. Due to the complexity of the systems to analyse, the mOSAIC Ontology is used in order to build modelling profiles in MetaMORP(h)OSY able to address cloud domain-related properties.

Key words: Cloud, Ontology, Model Driven Engineering, Validation And Verification.

1. Introduction. Cloud Computing is an emerging model for distributed systems. It refers both to applications delivered as services and to hardware, middleware and other software systems needed to provide them. Nowadays the Cloud is drawing the attention from the Information and Communication Technology (ICT) thanks to the appearance of a set of services with common characteristics which are provided by industrial vendors. Even if Cloud is a new concept, it is based upon several technologies and models which are not new and are built upon decades of research in virtualization, service oriented architecture, grid computing, utility computing or distributed computing ([16, 26, 43]). The variety of technologies and architectures makes the Cloud overall picture confusing [16]. Cloud service providers make resources accessible from Internet to users presenting them *as a service*. The computing resources (like processing units or data storages) are provided through virtualization. *Ad-hoc* systems can be built based on users requests and presented as services (*Infrastructure as a Service*, IaaS). An additional abstraction level is offered for supplying software platforms on virtualized infrastructure (*Platform as a Service*, PaaS). Finally software services can be executed on distributed platforms of the previous level (*Software as a Service*, SaaS). Except from these concepts, several definitions of Cloud Computing exist ([34, 5, 17, 13, 30, 25]), but each definition focuses only on particular aspects of the technology. Cloud computing can play a significant role in a variety of areas including innovations, virtual worlds, e-business, social networks, or search engines but it is actually still in its early stages, with consistent experimentation to come and standardization actions to effort. In this scenario, vendors provide different Cloud services at different levels usually providing their own interfaces to users and Application Programming Interfaces (APIs) to developers. This results in several problems for end-users that perform different operations for requesting Cloud services provided by different vendors, using different interfaces, languages and APIs. Since it is usually difficult to find providers which fully address all users needs, interoperability among services of different vendors is appealing.

Cloud computing solutions are currently used in settings where they have been developed without addressing a common programming model, open standard interfaces or adequate service level agreements or portability of applications. Neglecting these issues current Cloud computing forces people to be stranded into locked, proprietary systems. Developers making an effort in Cloudifying their applications cannot port them elsewhere.

In this scenario the mOSAIC project (EU FP7-ICT programme, project under grant #256910) aims at improving state of the art in Cloud computing by creating, promoting and exploiting an open-source Cloud application programming interface and a platform targeted for developing multi-Cloud oriented applications. One of the main goal is that of obtaining transparent and simple access to heterogeneous Cloud computing resources and to avoid locked-in proprietary solutions.

*Second University of Naples, Dep. of European and Mediterranean Studies, Via del Setificio 15, 81100 Caserta, Italy (francesco.moscato@unina2.it).

†Second University of Naples, Dep. of Information Engineering, Aversa, Italy

In order to attain this objective a common interface for users has to be designed and implemented, which should be able to wrap existing services, and also to enable intelligent service discovery. The keystone to fulfil this goal in mOSAIC is the definition of an ontology able to describe services and their (wrapped) interfaces.

In addition, users also require given qualities of services (QoS) and Cloud providers have to build services *on demand* depending on specified QoS. The provisioning should be coupled with proper monitoring systems which assure that services are provided with promised QoS also at run time.

In a Cloud scenario, Service Level Agreement (SLA) is a way to establish a contract between users and service providers where providers assure the execution of services with specified requirements. SLA is a way to express these requirements formally and formal methods can be exploited in order to verify them.

Automated service provisioning able to allocate and manage resources satisfying service goals is an open research challenge [46]. A methodology to analyse reachability of goal services with given requirement is appealing if it can be used to build automatically the target service.

Multi-agent systems(MAS) represent a model for designing and developing complex systems [6, 8, 47, 15] since it seems to cope with their increasing complexity. MAS can be successfully used to provide a model of Cloud System where several components and resources cooperate for providing complex services to users. Several methodologies have been proposed for MAS design and development [18, 9]. However software engineering has not provided yet any approach to model and verify their dependability during all the life cycle. Because of their criticality, Model Driven Engineering approaches are appealing when dealing with complex systems. Producing designs *correct by construction* where requirements are validated during all life cycle is useful. The methodology introduced here, MetaMORP(h)OSY, (Meta-modelling of Mas Object-based with Real-time specification in Project Of complex SYstems) inherits, improves and extends the one described in [28]. It is based on formal modeling and analysis of MAS systems. Cloud components for each service are modeled by using and extended UML profile compliant with MAS models. The main model is then analyzed by means of formal models that are obtained from the UML model with model transformation algorithms.

As explained below, the capability of representing Cloud services components and behaviours with UML-based diagrams is appealing since several services description and use cases are expressed by UML diagrams [36]. In addition, requested QoS may be represented as requirements that can be validated in a MDE methodology.

MetaMORP(h)OSY framework is based on Papyrus [41] and defines profiles for the definition of a modelling language for real-time MAS description. The language is compliant with OMG MARTE [12] specification, in order to make *MetaMORP(h)OSY* compliant with other tools supporting the standard. Verification at every life-cycle step is performed by implementing translation algorithms which translate design, simulation and run-time description into formal models.

Even if MetaMORP(h)OSY is used for (real-time) MAS modelling, it is based on a general methodology. When dealing with particular domains a way for generating modelling profiles based on the transformation of formal models describing the domain of interest is appealing. In mOSAIC, the Cloud ontology is the domain model that can be exploited in order to build missing information in MetaMORP(h)OSY allowing the framework for modelling and verifying requirements on Cloud components.

In particular, the intrinsic hierarchical organization of an ontology is useful when dealing only with particular aspects of the domain. In particular this work focuses on SLA modelling and verification of Cloud services. It will be shown how the mOSAIC Ontology can be used in order to build the part of the MetaMORP(h)OSY modelling profile required for mOSAIC components elements.

This paper is organized as follows: Section 2 shows some motivation for using mOSAIC Ontology for build a modelling profile in MetaMORP(h)OSY. Section 3 introduces the mOSAIC project, Section 4 contains a description of the mOSAIC Ontology; Section 5 describes MetaMORP(h)OSY modelling methodology and Section 6 reports a description of its modelling Profile and describes how mOSAIC Ontology is used in order to enhance it. Section 7 shows how the created profile can be exploited in modelling and verification of QoS of Cloud Services. Finally Section 8 contains some concluding remarks.

2. Motivation. mOSAIC promotes interoperability of cloud services. Building mOSAIC-compliant services requires the use of proper API and components. In addition, mOSAIC allows for the definition of SLA for services. SLA can be considered as requirements that have to be fulfilled at run-time by providers. Using Model Driven Engineering is appealing when dealing with complex cloud services with SLA since MDE allows services components to be built from design model definition. Anyway, the keystone to fulfil interoperability in mOSAIC is the definition of an ontology able to describe services and their (wrapped) interfaces. For searching and retrieving purposes, mOSAIC services have to be semantically annotated with elements of this ontology. This means that mOSAIC components and SLA definitions have to respect the organization and the structure of the ontology.

Hence an MDE methodology used to design and develop services in mOSAIC should use ontology information in its modelling profile in order to:

- inherit components that are defined as concepts in the ontology with their properties and relationships;
- allow for specification of QoS parameters;
- semantically annotate services components created with the MDE framework.

Using ontology information in a MDE modelling profile is appealing since MDE model transformation techniques enable automatic creation of mOSAIC components and interfaces. In addition, formal verification in MDE assures, at least on the design model, that QoS requirements are respected.

In addition, if proper model transformation techniques are used (like in MetaMORP(h)OSY), verification can also be enacted at run-time by monitors that are created automatically.

In MetaMORP(h)OSY, the mOSAIC ontology has been used to generate part of a modelling profile that is able to describe cloud components in a vision compliant with mOSAIC architecture.

3. mOSAIC Project. The Open Cloud Manifesto [39] identifies five main challenges for Cloud: data and application interoperability; data and application portability; governance and management; metering and monitoring; security.

Actually, the main problem in Cloud computing is the lack of unified standards. Market needs drive commercial vendors to offer Cloud services with their own interfaces since no standards were available at the moment. Vendors solutions have arisen as commonly used interface for Cloud services but interoperability remains an hard challenge, like portability of developed services on different platforms. In addition vendors and open Cloud initiatives spent few efforts in offering services with negotiated quality level.

The mOSAIC project tries to fully address the first two challenges and partially addresses the next two ones by providing a platform which:

- enables interoperability among different Cloud services,
- eases the portability of developed services on different platforms,
- enables intelligent discovery of services,
- enables services composition,
- allows for management of Service Levels Agreement (SLA).

The architecture of mOSAIC platform is depicted in Fig.3.1:

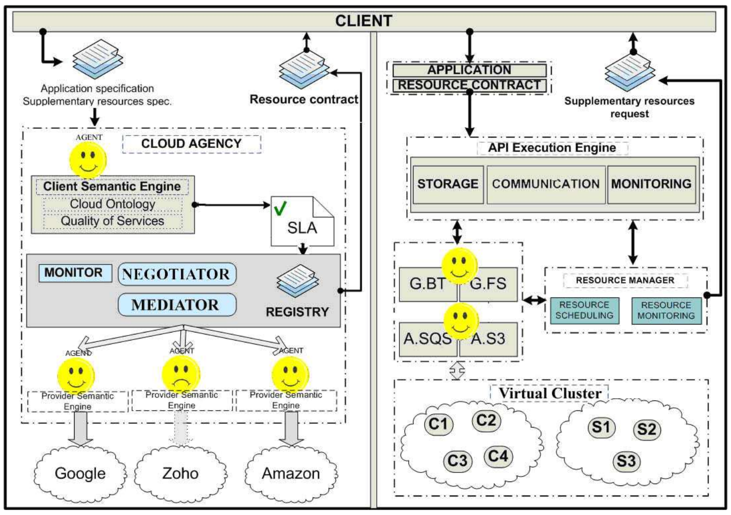


Fig. 3.1: mOSAIC Architecture

it provides facilities both for end-users (at the left of Fig.3.1) and for services developers and managers (depicted on the right side of Fig.3.1)

From the end-users' point of view, the main component is the *Cloud Agency.* This consists in a core set of software agents which implement the basic services of this component. They include:

- negotiation of SLAs;
- deployment of Cloud services;
- discovery and brokering of Cloud services.

In particular, *Client Agent* is responsible for collecting users' application requirements, for creating and updating the SLAs in order to grant always to best QoS. The *Negotiator* manages SLAs and mediates between the user and the broker; it selects protocols for agreements, negotiates SLA creation, and it handles fulfilment and violation. The *Mediator* selects vendor agents able to deploy services with the specified user requirements; it also interfaces with services deployed on different vendors' providers. The *Provider Agent* interacts with virtual or physical resources at provider side. In mOSAIC the Cloud Agency was built upon the MAGDA [2] toolset, which provides all the facilities to design, develop and deploy agent-based services. The *semantic engine* uses information in the Cloud Ontology to implement a semantic-based Cloud services discovery exploiting semantic, syntactic and structural schema matching for searches.

In the Cloud developers and managers perspective, the main components of mOSAIC Architecture are the *API execution engine* and the *Resource Manager.* The first one offers a unique API to use Cloud Services from different vendors when using and developing other services. The API execution engine is able to wrap storage, communication and monitoring features of Cloud platforms. In particular, Virtual Clusters (VC) [11] are used as resource management facility. They are configured by software agents in order to let users to configure required services. A *Resource contract* will grant user's requirements and the Resource Manager will assign physical resources to VC on the basis of the contract.

In this architecture, the bonding element which allows for interoperability and resources description is the *Cloud Ontology.* It is the base for Cloud services and resources description and it contains all information needed to characterize API also from a semantic point of view.

The Cloud Ontology is based on several Cloud taxonomies proposed in literature [1, 38, 19, 7, 20]. It is developed in OWL [24] and OWL-S languages [22]. The benefit of using an ontology language is that it acts as a general method for the conceptual description or modelling of information that is implemented by actual resources [37]. mOSAIC aims at developing ontologies that would offer the main building block to describe services at the three delivery models of Cloud Computing (i.e. IaaS, PaaS, SaaS).

4. mOSAIC Ontology. Ontologies offer the means of explicit representation of the meaning of different terms or concepts, together with their relationships. They are directed to represent semantic information, instead of content. Different languages can be considered for the specification of ontologies, including DAML, OIL, RDF and RDFS, OWL or WSML.

The Web Ontology Language (OWL) is a standard from [24, 4], based on XML, RDF and RDFS. With OWL complex relationships and constraints can be represented in ontologies. With important revisions to the language, OWL 2 became the W3C recommendation in 2009, introducing features to improve scalability in applications. [14]

Different efforts to formalize Semantic Web developments exist. Web Service Modeling Ontology (WSMO) [42] "*provides the conceptual underpinning and a formal language for semantically describing all relevant aspects of Web services in order to facilitate the automatization of discovering, combining and invoking electronic services over the Web*" [33]. WSML was offered as a companion language to WSMO, for representing modelled ontologies by a common terminology for Web Services interactions [10, 33]. The Semantic Web Services Framework (SWSF) offers a similar approach, with its two major components, the Semantic Web Services Language (SWSL) and the Semantic Web Services Ontology (SWSO) [3].

Semantically-enabled services offer the means for intelligent selection of services, with automation of different tasks, including service discovery, mediation, invocation, or composition. Current research efforts are enhancing typical web services technologies in order to provide a semantically-enhanced behaviour in developments like OWL-S [22], WSDL-S and METEOR-S [32, 29], WSML [10], WSMO [33], or SWSF [3].

The top level of the mOSAIC Ontology is shown in Fig.4.1 which reports the main concepts of the mOSAIC ontology. Concepts have been identified analysing standards and proposals from literature. In the following its main concepts will be listed and described. A deeper description of the mOSAIC ontology is in [27], this works describes only the elements which are used for the creation of the modelling profile in MetaMORP(h)OSY.

The **Language** class contains instances of languages used for APIs implementation (for example, Java and

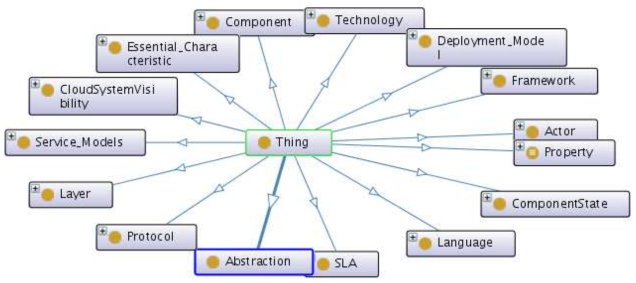


Fig. 4.1: Top Level Concepts in mOSAIC Ontology

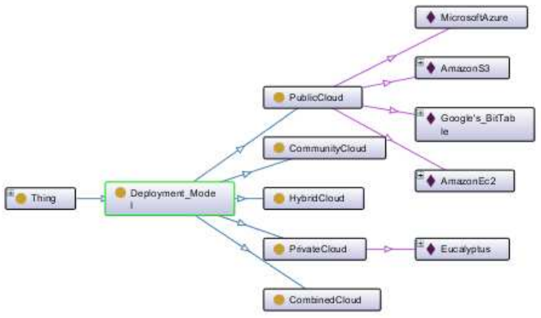


Fig. 4.2: Deployment Model

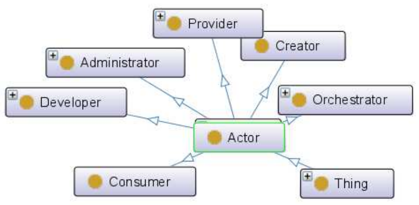
Python). **Abstraction** class contains the abstraction level at which services are provided as described in [45]. Here, Cloud services belong to the same layer if they have equivalent level of abstraction. **Deployment Model** class includes concepts required by Cloud NIST [45] standard for what deployment model of Cloud services concerns. **Essential Characteristics** class includes individuals which are defined by NIST. **Framework** class contains individuals that identify programming framework supporting API programming Languages. **Actor** contains subclasses where actors interacting with Cloud systems are divided. **Property** subclasses contain all elements needed for describing characteristics of Cloud resources. These are also used to specify SLA requirements. **ComponentState** includes all concepts for defining the states which Cloud components and resources may assume. **SLA** class defines concepts for SLA definitions. **Protocol** class contains individuals for protocols used in communication among Cloud components. **Layers** class distinguishes firmware, hardware and software infrastructures for Cloud platforms. **Service Models** class includes all kinds of services provided by Cloud Systems. **Predicate** contains classes used for description of the behaviours of statefull Cloud components. **CloudSystemVisibility** class allows for specification of Cloud systems visibility, like private and public clouds. **Component** is the main class of mOSAIC ontology. All cloud elements (resources, services, infrastructures etc.) are its subclasses. **Technology** class contains all concepts related to technology involved in Cloud services provisioning, like virtualization.

Fig.4.2 shows the *Deployment_Model* subclasses.

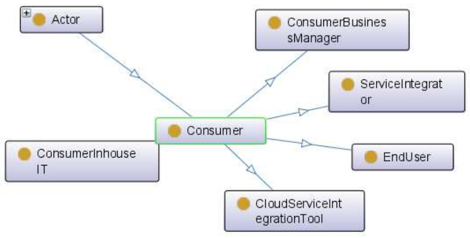
They include several types of deployment models for Cloud Systems: PublicCloud contains all individuals providing public or world wide access to their resources, like MicrosoftAzure, Amazon and Google. PrivateCloud instead is related to Deployment Models of framework that can provide access to private Cloud resources, like Eucaliptus.

The Actor class identifies cloud actors, that can be divided as in Fig.4.3.

Provider, Consumer and Creator subclasses follow the IBM cloud computing reference Architecture [23]. Administrator manages cloud infrastructure; Orchestrator composes Cloud Services in order to provide value



(a) Actor

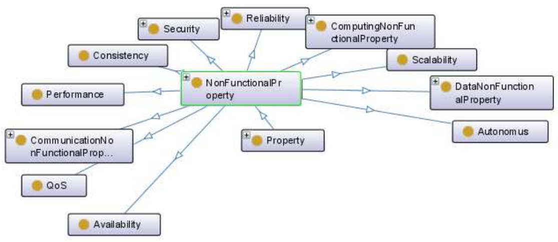


(b) Consumer

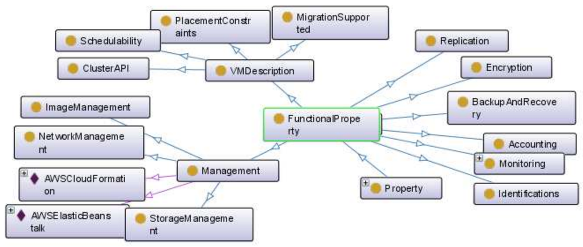
Fig. 4.3: Actors

added services; Developer implements new Cloud Services. Notice that the difference between Developer and Creator, is in the way they interact with cloud Providers. Developers use offline resources (tools and frameworks) in order to implement new Cloud Service. A Creator instead builds cloud services by using functionalities exposed by a Cloud Service Provider. Consumer Actors can be further divided as shown in Fig. 4.3(b).

Some Property's subclasses are shown in Fig.4.4. They are divided into NonFunctionalProperties and FunctionalProperties that respectively define the sets of non functional and functional properties of a Cloud Component. Properties can be used to characterize Cloud Components (services, infrastructure etc.) and to request given characteristics for components when dealing with SLA.



(a) Non Functional Properties



(b) Functional Properties

Fig. 4.4: Properties

The main non-functional properties for cloud components are: Scalability; Autonomy; Availability; QoS; Performance; Consistency; Security; Reliability.

Computing Non Functional properties can be divided into CPU and Memory related properties. A deeper division identifies: CPUSpeedProperty; CPUNumberOfCores; CPUArchitecture; CPUTypeProperty and CPU-FlopsProperty. These properties are used to specify the clock frequency, the number of cores, the architecture, the model and the FLOPS of CPUs respectively. The properties follow the OCCI [40] standard and API. A Data Property is defined for each of them in order to specify the value of the property for the related individuals. Properties for memory are divided into: MemoryAllocationProperty and MemorySize. The first property is used to specify memory allocation policies while the second one is used to declare (or require) the amount of memory in a Cloud infrastructure.

Subclasses of this Network Non Functional element are: NetworkLatencyProperty, NetworkDelayProperty, NetworkBandwidthProperty. The first class is used to define the mean latency of a network, the second one the mean, the maximum and the minimum delay for packets and the last one is used to define the mean and the maximum bandwidth of a network. The values for individuals are defined by specifying proper data properties

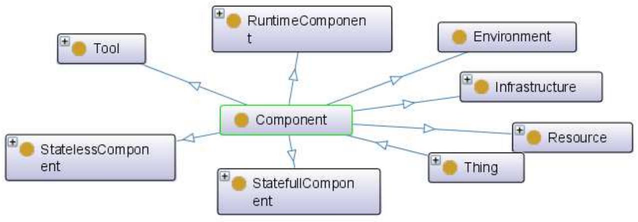


Fig. 4.5: Component

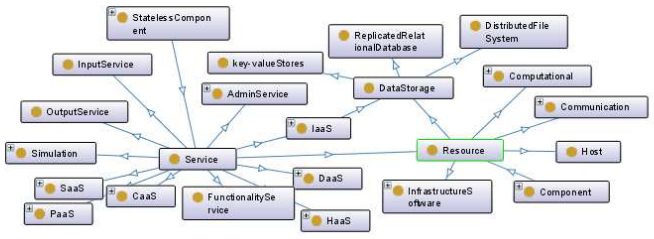


Fig. 4.6: Resource and Services

defined on these classes. Data non functional properties are related to disk size (DiskSpaceProperty), transfer rate (DiskTransferRateProperty) and bandwidth (DiskBandwidthProperty).

The main functional properties are: Replication (for the definition of the type of replication policies of resources); Encryption (it specifies the encryption policies of resources); BackupAndRecovery (it is used to describe the back up and recovery strategies used for a Cloud Component); Accounting (its individuals define the accounting policies for resources); Monitoring (this class allows for the specification of monitoring policies for resources); Identification (it contains individuals that can specify the algorithms and policies for users identification); VMDescription (used to describe virtual machines technologies and configuration eventually used in cloud infrastructure); Management (it defines the management policies for cloud resources).

Management contains the following subclasses: ImageManagement, NetworkManagement and StorageManagement. The first one is used to define the management policies of a VM image, the second one to define network management policies in a cloud infrastructure, while the third one is used to define storage management policies for cloud resources.

Service_Models subclasses includes all models for services in Cloud. Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Service as a Service (SaaS) are the classical models defined in the NIST standard, while the last, BPaaS (Business Process as a Service) is defined in the IBM Cloud Computing Reference Architecture.

Component Subclasses are reported in Fig.4.5.

They are divided into: Tool (this contains all tools used for Cloud services development or cloud resources management); RunTimeComponent (this class contains the elements for defining mOSAIC run-time components); Environment (used to define individuals concerning the cloud environment used by cloud services); Infrastructure (describes the component in the cloud infrastructure); StatefullComponent and StatelessComponent; Resource (it collects all resource classes in the cloud ontology).

The Resource class is the most complex in mOSAIC, since, following the OCCI documentation, in Cloud Systems everything is a cloud Resource. Hence this is a super class for other main cloud components as shown in Fig.4.6.

Service is a Resource. Platform as a Service (PaaS), Computing as a Service (CaaS), Data as a Service (DaaS), infrastructure as a Service (IaaS), Hardware as a Service (HaaS) and other service models (Simulation, services which offers some functionalities, admin, data input and output services) are subclasses of Services. For example, in Figure, DataStorage is provided as an IaaS. Key-valueStores, ReplicatedRelationalDatabeses and DistributedFileSystems are examples of DataStorage services. Cloud Component are also considered as Cloud Resources, like Hosts, Computational and Communication resources or InfrastructureSoftware.

5. MetaMORP(h)OSY modelling methodology. The *MetaMORP(h)OSY* modelling methodology (MMM) extends and improves the one used in the REMM [28] framework. In REMM, systems are modelled as Multi Agents, by using Beliefs, Desires, Intentions (BDI) logics [44]. REMM models are based on a UML modelling profile that implements a modelling language (RT-AML) for real-time BDI MAS. Requirements are verified at design phase by using formal methods following a MDE approach. REMM models are translated into timed automata in order to check properties expressed in timed temporal logics. At run-time, REMM provides a mean for checking properties verified at design-time. This is achieved by collecting run-time measures that are used to tune again design models. They are then used to verify previously checked properties in the case temporal behaviour of real systems differs from the designed one. REMM framework also provides translation from design models to Jadex [31] run-time agents with real-time scheduling features.

The REMM methodology has some limitation. First of all, requirements can be expressed only in timed temporal logics and the only way to analyse them is to translate models into timed automata. At design phase, no way is provided for choosing the type of analysis required on the model. In addition recently OMG has defined a UML profile for modelling real-time systems (MARTE [12]). Since the methodology on which REMM (and *MetaMORP(h)OSY*) is based on UML models, the use of a profile which also supports MARTE standard is appealing. Finally, REMM does not provide any translations features for producing simulation models. Simulation gives the ability of performing fault injection and other fault analyses on the system to implement, and this is useful especially for critical systems.

The MMM overcomes these problems, extending and improving the modelling language, the profile used in REMM, and the techniques used to translate modelling and simulation models.

The main phases of the MMM are depicted in Fig.5.1.

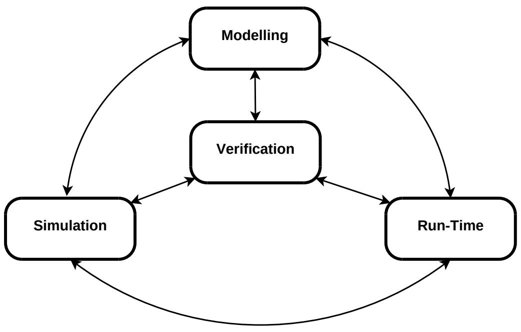


Fig. 5.1: *MetaMORP(h)OSY* methodology

They consist in: *Modelling*, *Simulation*, *Run-time* and *Verification*. They will be described briefly in the following.

5.1. Modelling phase. In this phase design models for the system to study and realize, and requirements to verify on the system are defined. Models are described by using the Real-Time Agent Modelling Language (RT-AML) which extends the one presented in [28]. RT-AML is based on UML diagrams, and extensions of Class diagrams, Activity diagrams and Sequence diagrams are used to describe the BDI model of agents. Models developed in this phase can be used to define run-time and simulation components. According to MDE philosophy, requirements have to be verified on models even at this stage. Proper translation techniques are defined to build models used in verification phase. Models generated from verification depend on requirements to verify and on the analysis to perform.

5.2. Simulation phase. In this phase Simulation models are defined or generated automatically from models defined in modelling phase. In the last case, stubs are generated from agent diagrams and real-time schedulers and monitors are provided in order to verify requirements. The simulation models used in MMM are based on MASON [21] toolkit and can be augmented with fault-injection by using proper libraries. Requirements verification is instead performed at verification stage, by using formal models which have been eventually generated during the modelling phase.

Agents behaviours during simulation may differ from the ones defined in the modelling phase. This happens when faults are injected in the simulation, or if conditions that may change temporal behaviours of the agents are considered in simulation. In these cases, requirements defined in simulation phase can be verified on the same models used for verification in modelling phase. The difference is in the parameters used to tune the verification models, which are collected from simulated behaviour.

5.3. Run-Time phase. In this phase the real system is developed and executed. Stubs for run-time components can be generated from modelling phase as for simulation. Also verification models can be used in real-time scheduling in order to forecast if real-time constraints and other requirements are verified at run-time, in the same way of simulation. In addition, code from simulation can be used to implement the system, and also simulation models can be used at run-time for verification purposes.

5.4. Verification. Verification phase is driven by formal models. Models from modelling phase are translated into models which are useful for verifying system requirements. The models generated for verification depends on the properties and on the analyses to perform.

In order to model agents and their interactions in MetaMORP(h)OSY, three kinds of diagrams have to be provided: an *Agent Diagram*, *Activity Diagrams* for agents plans, and a *Sequence Diagram*.

In the *Agent Diagram* the MAS system structure is described. Here classes with proper stereotype represent Agents with their Plans and Beliefs.

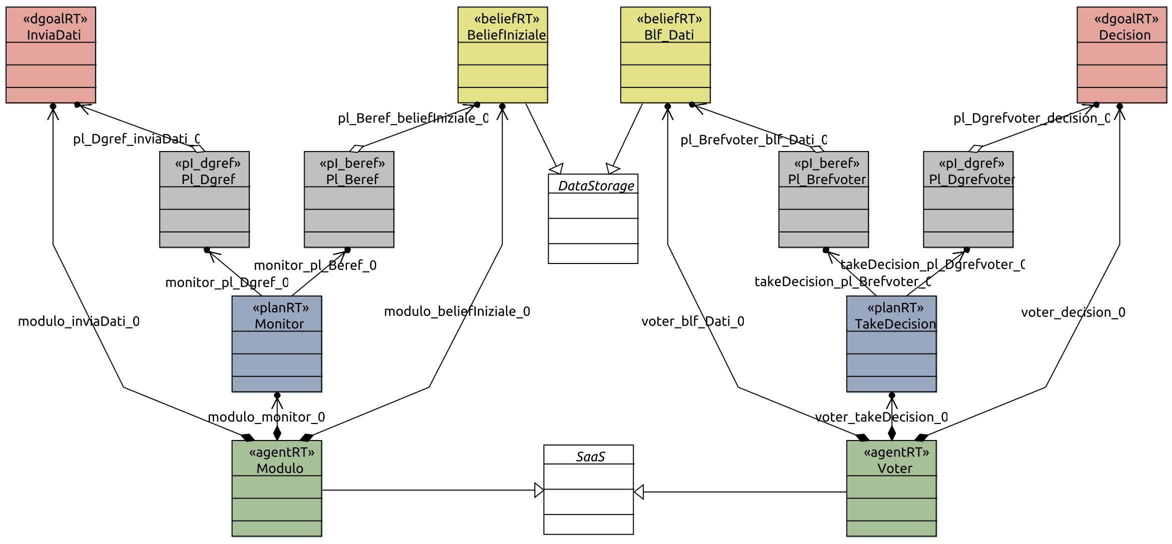


Fig. 5.2: Agent Diagram

For example, diagram in Fig.5.2 depicts two agents. The stereotype *AgentRT* is applied to these classes. It is defined in the MetaMORP(h)OSY modelling profile and it is used to define properties that can specify (real-time) agents in a UML model. By associating this stereotype to a class in a MetaMORP(h)OSY model, it is possible to specify agent's temporal characteristics. *PlanRT*, *BeliefRT* and *DGoalRT* stereotypes are defined in the MetaMORP(h)OSY modelling profile which in turn are used to define plans, beliefs and goals of agents. Plans are related to some beliefs and pursue some goals. Relationships between plans, beliefs and goals are implemented by mean of other stereotypes which description is omitted for brevity.

The Agent Diagram in Fig.5.2 describes agents structures, listing their plans, beliefs and goals related to each plan. In order to complete the model a dynamic description of agents behaviours has to be provided. In MetaMORP(h)OSY this is done by mean of particular activity and sequence diagrams.

MetaMORP(h)OSY profile defines several stereotypes for messages and timed activities in the activity diagrams. This allows for the analysis of timed behaviour and interaction of agents. At the state, each PlanRT in the Agent Diagram is associated to an Activity diagram which describes the action enacted during plan execution. Each action is described in terms of expected execution time, messages awaited from and sent to other agents, resources and beliefs involved in the action.

For example, in Fig.5.3 the Activity diagram for the *TakeDecision* plan is shown.

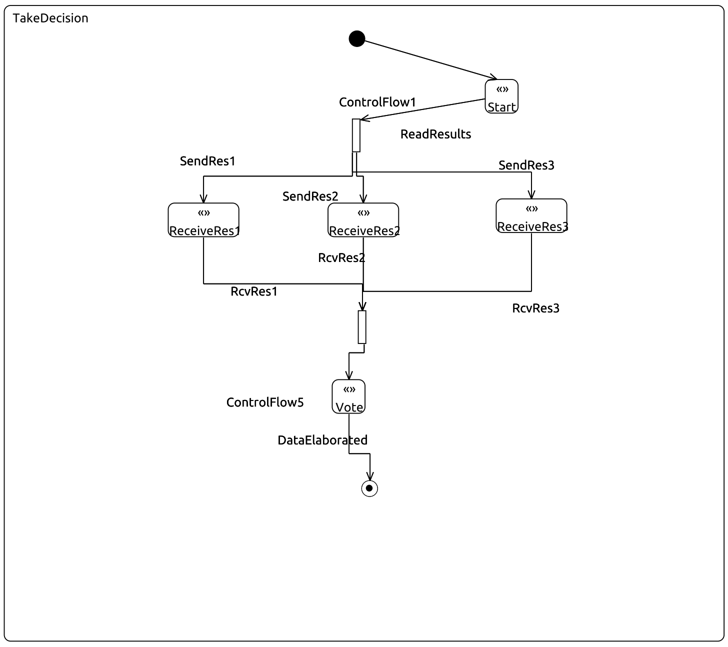


Fig. 5.3: TakeDecision Plan

Activity diagrams describe agents behaviours in regard to their plans, but different execution paths are possible depending on different use cases. In order to define which events and messages are involved in a particular use case, a Sequence Diagram is used in MetaMORP(h)OSY. The main purpose of this diagram is the definition of agents interactions in a use case.

All messages in the sequence are *StimulusRT*: a stereotype able to represent the timed behaviour of messages exchanged during the execution of a use case. *StimulusRT* messages are related to messages sent and received in activity diagrams.

Fig.5.4 depicts a Sequence Diagram.

6. MetaMORP(h)OSY modelling profile. MetaMORP(h)OSY modelling methodology is based on the creation of a UML modelling profile, that can be considered as a meta-language for definition of components in UML models.

The basic MMM Modelling profile extends the MARTE [12] profile. It defines the stereotypes and the properties needed to define a MAS system. Fig.6.1 shows the profile used for Agent Diagrams definition.

The Profile extends the basic UML and MARTE profiles. This means that classical UML elements and stereotypes from MARTE meta-language can be used in a model compliant with the profile. In addition, it introduces the elements in Tab.6.1

Agent Plans are modelled by using an extension of the UML Activity diagram. Fig.6.2 shows the extended profile. It extends the basic UML activity diagrams with the elements described in Tab.6.2

Activity diagrams defined with the previous elements take into account of interaction among agents. In order to cope with complexity, it is possible to define the sequence of messages that agents exchange during the execution of a particular use case. Fig.6.3 depicts the elements in the MetaMORP(h)OSY profile used for sequence diagrams definition.

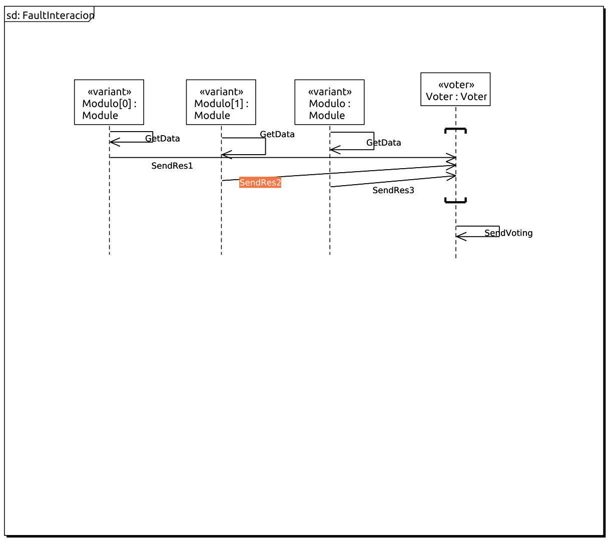


Fig. 5.4: Sequence Diagram

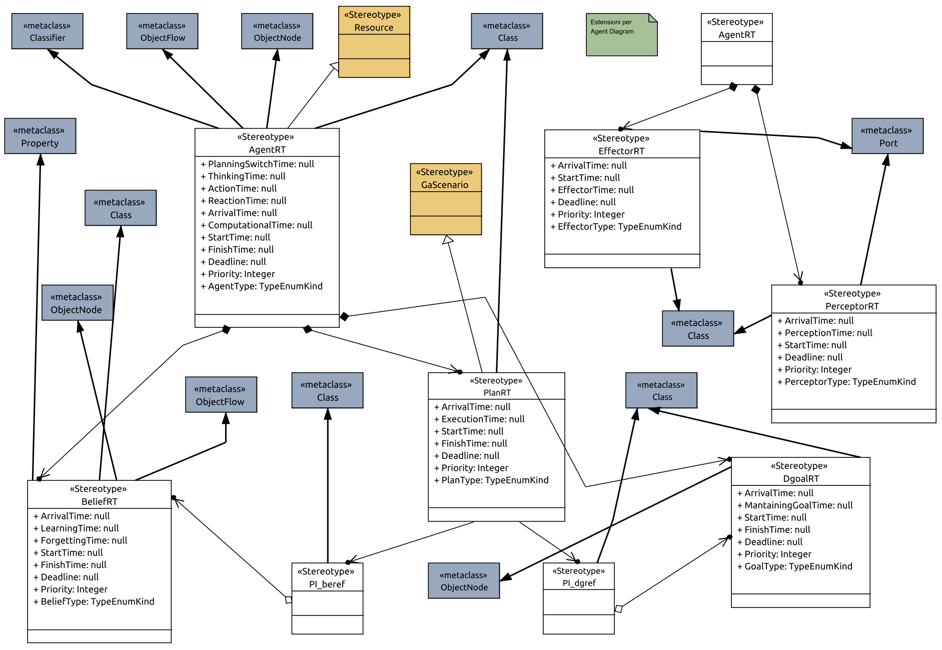


Fig. 6.1: Agent Diagram Profile

Again the profile extends the basic UML sequence diagrams and allows for the definition of the elements listed in Tab.6.3.

The basic modelling profile of the MMM is used in order to describe MAS structure and behaviours, but it lacks of detailed information about modelling domain. This information can be retrieved from the mOSAIC ontology. The ontology is translated into a set of classes that are inherited during Agent Diagram modelling phase. The translation first identifies equivalent classes in the ontology. The ontology taxonomy is then translated into a hierarchy of UML classes and then relationships among classes are translated into associations.

Part of the hierarchy produced from the piece of the ontology in Fig.4.6 is depicted in Fig.6.4.

Table 6.1: Agent Diagram Elements

Component	Description
AgentRT	Used for definition of agents with temporal description
PlanRT	Used for the definition of agent plans. Each plan is related to an Activity diagram that reports the behaviour of the agent while following the plan in terms of Action States
DGoalRT	Used for the definition of agents goals. Agents pursue goals while executing plans
PerceptorRT and EffectorRT	These are commonly used to define inputs and outputs for agents
BeliefRT	Beliefs are used to store status for agents containing their beliefs about the external environment and other agents

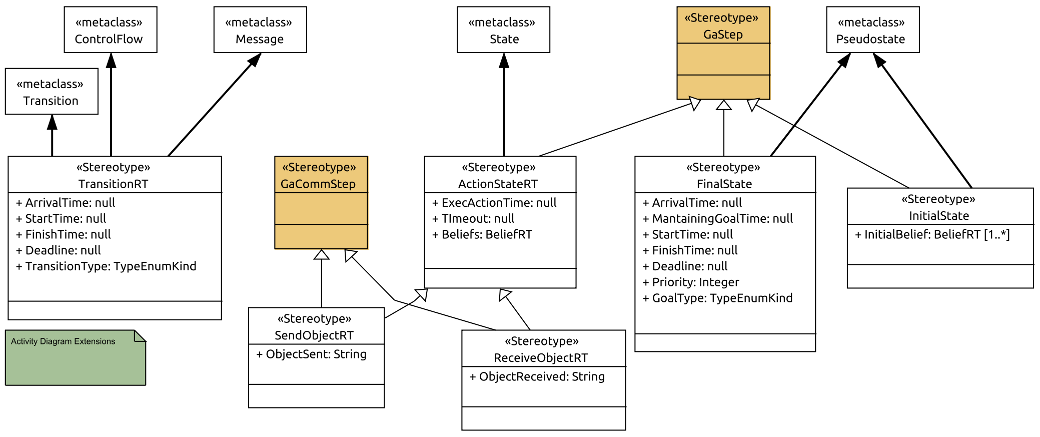


Fig. 6.2: Activity Diagram Profile

Following the MARTE specification, properties (and requirements) on a model can be analysed or monitored by elements called *Observer* [12].

Verification in MetaMORP(h)OSY is enacted by means of model transformation and model analysis. For example, state reachability of goals under real-time constraints is performed by translating the RT-AML model into Timed Automata and then executing a model checker on the translated model([28]).

In order to specify on which elements Observers work, it is necessary to identify the elements on which properties can be evaluated. For example, (timed) state reachability analysis is available on DGoalRT elements, that are associated to final states of activities.

When dealing with complex SLA, the generation of the profile containing properties definition is appealing. The mOSAIC ontology is used in order to generate properties modelling profile.

Thanks to the hierarchical organization of elements in the ontology, subclasses of *Property* concept (see Fig.4.4) are identified and inserted in the modelling profile as shown in Fig. 6.5

The ontology also contains relationships containing information about the classes on which properties can be defined. For example, Availability can be requested on storage resources etc. This information is used in order to complete the properties profile and to define proper Observers.

7. Example. In this section an example of the use of MetaMORP(h)OSY for definition of a cloud service with a requested SLA. It will be shown how MMM is used in order to validate the SLA on the model *before* user and provider agree on the offered service.

In the example the user requests a service with high availability. The service vendor is able to provide a service with triple redundancy and voting and it want to assure that its composed service has the level of reliability required by the user. The service (called here simply *Module*) is replicated thrice and each service works stand-alone producing its own results. Results are than collected by a service called *Voter* that forward results only if two on three are equal.

The Agent Diagram of the system is reported in Fig.5.2 where the two agents representing the *Module* and *Voter* services are depicted. The Module agent has the goal of sending computed results to the Voter (*SendData*)

Table 6.2: Activity Diagram Elements

Component	Description
ActionStateRT	Actions in agents plan are modelled with this component that allows for specification of temporal behaviour of actions
TransitionRT	Transitions among ActionStateRT. For these elements it is possible to specify events synchronizations and deadlines
InitialState FinalState	Initial and Final states of the plan usually final state can be associated to a DGoalRT

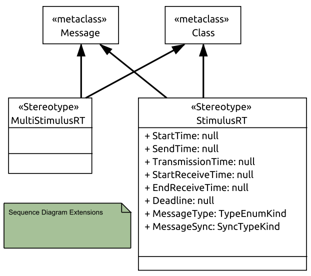


Fig. 6.3: Sequence Diagram Profile

and the Voter agent has the goal of taking a decision (*Decision*).

The *SendData* goal is achieved by executing the *Monitor* plan, while *Decision* by executing the *TakeDecision* plan. Both Module and Voter are provided as *SaaS*. And the beliefs of the agents are stored in *DataStorage*. For each Agent properties defined in the MetaMORP(h)OSY profile are used. In particular, in this example, each Agent has its own Fault probability and its own Mean Time to Failure that will be used in order to evaluate global reliability on the system.

The *Monitor* plan is simple and it is not described for brevity.

The *TakeDecision* plan is shown in Fig.5.3, where the voter requests results from the three modules ($Receive\_Res(i)$) and *vote*s when messages arrive.

The Sequence Diagram in Fig.5.4 is used to define that three different *Module* agents send their messages to enact voting. Voting agent collects messages requiring that at least 2 messages correctly arrive. StimulusRT stereotype is used for messages and information about messages reliability is reported in their modelling properties.

Finally, an Observer for the evaluation of the global reliability of the modelled system is defined. This is depicted in Fig.7.1

The Observer translates the RT-AML model into a Fault Tree Model and generates the input for the Sharpe [35] framework in order to evaluate availability, but the description of the analysis model is out of the scope of this work. Service providers can change model parameters in order to establish the correct configuration of services for assuring the requested QoS.

8. Conclusions. In this paper the mOSAIC Ontology, the MetaMORP(h)OSY methodology and framework have been introduced. It has been shown how domain-related information contained in the ontology can be used in order to enhance a modelling profile for formal verification of QoS of Cloud service. It has been shown how MetaMORP(h)OSY suites well the MAS nature of cloud components and it is possible to define Observers on models for system analysis. Future works include the design and the analysis of high available and fault tolerant scenarios in the mOSAIC project.

Acknowledgments. This work has been supported by the mOSAIC project (EU FP7-ICT programme, project under grant #256910).

REFERENCES

Table 6.3: Sequence Diagram Elements

Component	Description
StimulusRT	This elements allows for the specification of asynchronous and synchronous messages, where deadline, arrival time and other temporal properties can be specified
MultiStimulusRT	It is a StimulusRT for redundant messages

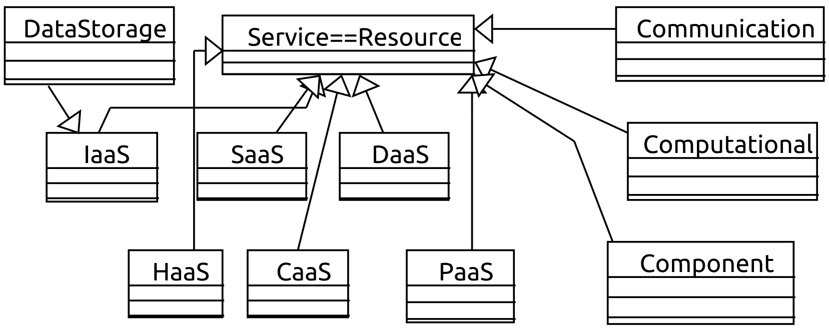


Fig. 6.4: Classes imported from mOSAIC ontology

[1] Appistry, *Cloud Taxonomy: Applications, Platform, Infrastructure : http://www.appistry.com/blogs/sam/cloud-taxonomy-applications-platform-infrastructure*, 2008.

[2] R. Aversa, B. Di Martino, N. Mazzocca, and S. Venticinque, *A skeleton based programming paradigm for mobile multi-agents on distributed systems and its realization within the magda mobile agents platform*, Mob. Inf. Syst., 4 (2008), pp. 131–146.

[3] S. Battle, A. Bernstein, H. Boley, B. Grosof, M. Gruninger, R. Hull, M. Kifer, D. Martin, S. McIlraith, D. McGuinness, J. Su, and S. Tabet, *Semantic web services framework*, September 2005.

[4] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. Mcguinness, P. F. Patel-Schneider, and L. A. Stein, *Owl web ontology language reference*, tech. rep., W3C, 2004.

[5] R. Buyya, C. S. Yeo, and S. Venugopal, *Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities*, in HPCC '08: Proceedings of the 2008 10th IEEE International Conference on High Performance Computing and Communications, IEEE Computer Society, September 2008, pp. 5–13.

[6] B. Chen, H. H. Cheng, and J. Palen, *Integrating mobile agent technology with multi-agent systems for distributed traffic detection and management systems*, Transportation Research Part C: Emerging Technologies, 17 (2009), pp. 1 – 10.

[7] C.Hoff, *Cloud Taxonomy and Ontology : http://rationalsecurity.typepad.com/ blog/2009/01/cloud-computing-taxonomy-ontology.html*, 2009.

[8] M. Cossentino, P. Burrafato, S. Lombardo, and L. Sabatucci, *Introducing pattern reuse in the design of multi-agent systems*, in Agent Technologies, Infrastructures, Tools, and Applications for E-Services, vol. 2592 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2003, pp. 107–120.

[9] V. T. Da Silva and C. J. P. De Lucena, *From a conceptual framework for agents and objects to a multi-agent system modeling language*, Autonomous Agents and Multi-Agent Systems, 9 (2004), pp. 145–189.

[10] J. De Bruijn, H. Lausen, R. Krummenacher, A. Polleres, L. Predoiu, M. Kifer, and D. Fensel, *The web service modeling language wsml*, tech. rep., DERI, October 2005.

[11] U. V. E.P. Mancini, M. Rak, *PerfCloud: GRID Services for Performance-oriented Development of Cloud Computing Applications*, in Proceedings of WETICE, IEEE Computer Society, July 2009.

[12] M. Faugere, T. Bourbeau, R. de Simone, and S. Gerard, *Marte: Also an uml profile for modeling aadl applications*, Engineering of Complex Computer Systems, IEEE International Conference on, 0 (2007), pp. 359–364.

[13] Galen Gruman and Eric Knorr, *What cloud computing really means. InfoWorld : http://www.infoworld.com/article/08/04/07/15FE-cloud-computing-reality 1.html*, 2008.

[14] B. C. Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider, and U. Sattler, *Owl 2: The next step for owl*, Web Semant., 6 (2008), pp. 309–322.

[15] Z. Guessoum, J.-P. Briot, N. Faci, and O. Marin, *Towards reliable multi-agent systems: An adaptive replication mechanism*, Multiagent Grid Syst., 6 (2010), pp. 1–24.

[16] K. Hwang, *Massively distributed systems: From grids and p2p to clouds*, in Proceedings of The 3rd International Conference on Grid and Pervasive Computing - gpc-workshops, 2008, p. xxii.

[17] Jeremy Geelan, *Twenty one experts define cloud computing. Virtualization : http://virtualization.sys-con.com/node/612375*, 2008.

[18] K. M. Kavi, M. Aborizka, D. Kung, and N. Texas, *A framework for designing, modeling and analyzing agent based software systems*, in in Proc. of 5th International Conference on Algorithms and Architectures for Parallel Processing, 2002, pp. 23–25.

[19] P. Lairds, *Cloud Computing Taxonomy*, in Procs. Interop09, IEEE Computer Society, May 2009, pp. 201–206.

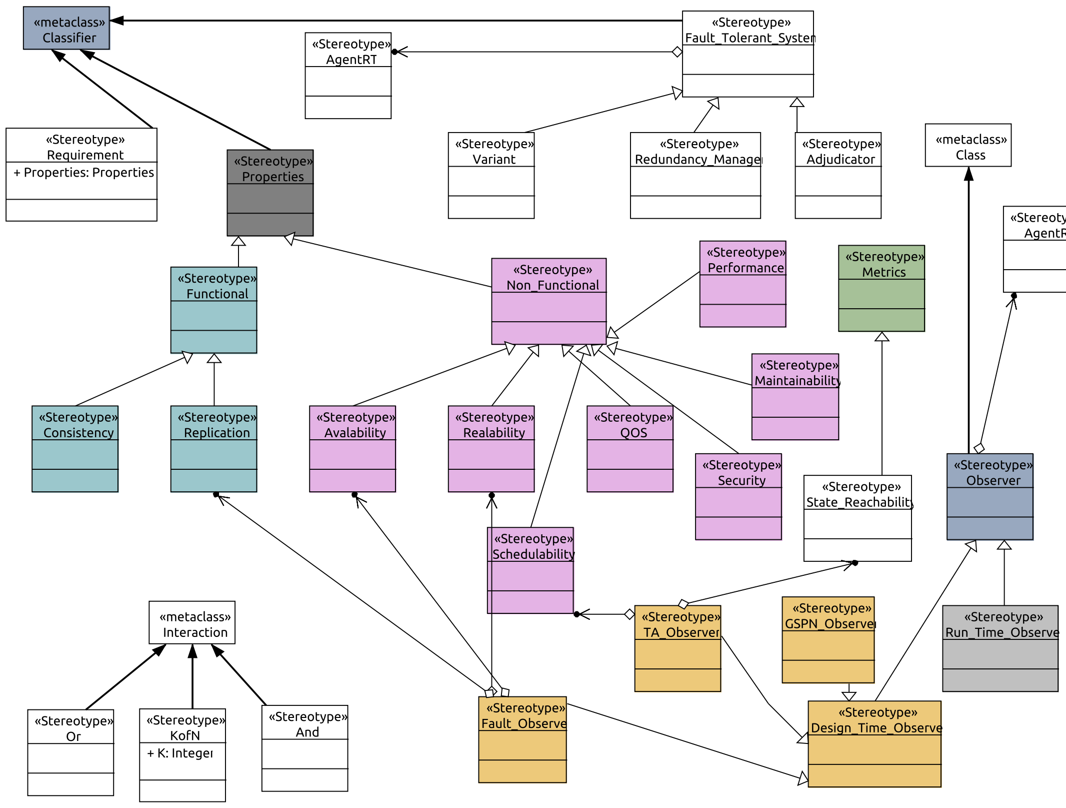


Fig. 6.5: Properties Diagram Profile

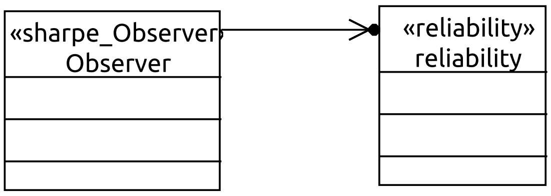


Fig. 7.1: Reliability Observer

[20] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm, *What's inside the cloud? an architectural map of the cloud landscape*, in Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing, CLOUD '09, Washington, DC, USA, 2009, IEEE Computer Society, pp. 23–31.
[21] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. C. Balan, *Mason: A multi-agent simulation environment*, Simulation, 81 (2005), pp. 517–527.
[22] D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, N. Srinivasan, and K. Sycara, *Bringing Semantics to Web Services: The OWL-S Approach*, in SWSWPC 2004, J. Cardoso and A. Sheth, eds., vol. 3387 of LNCS, Springer, 2004, pp. 26–42.
[23] M.Behrendt, B.Glasner, P.Kopp, R.Dieckmann, G.Breiter, S.Pappe, H.Kreger, and A. Arsanjani, *Introduction and Architecture Overview, IBM Cloud Computing Reference Architecture 2.0: https://www.opengroup.org/cloudcomputing/uploads/40/23840/CCRA.IBMSubmission.02282011.doc*, 2011.
[24] McGuinness, D.L., van Harmelen, F., *OWL Web Ontology Language Overview. W3C Recommendation: http://www.w3.org/TR/2004/REC-owl-features-20040210/*, 2004.
[25] Members of EGEE-II, *An egee comparative study: Grids and clouds - evolution or revolution. Technical report, Enabling Grids for E-sciencE Project : https://edms.cern.ch/document/925013/*, 2008.
[26] D. Milojicic, *Cloud computing: Interview with russ daniels and franco travostino*, IEEE Internet Computing, (2008), pp. 7–9.
[27] F. Moscato, R. Aversa, B. Di Martino, T. Fortis, and V. Munteanu, *An analysis of mosaic ontology for cloud resources annotation*, in Computer Science and Information Systems (FedCSIS), 2011 Federated Conference on, IEEE,

2011, pp. 973–980.
[28] F. Moscato, S. Venticinque, R. Aversa, and B. Di Martino, *Formal modeling and verification of real-time multi-agent systems: The remm framework*, in Intelligent Distributed Computing, Systems and Applications, C. Badica, G. Mangioni, V. Carchiolo, and D. Burdescu, eds., vol. 162 of Studies in Computational Intelligence, Springer Berlin / Heidelberg, 2008, pp. 187–196.
[29] A. A. Patil, S. A. Oundhakar, A. P. Sheth, and K. Verma, *Meteor-s web service annotation framework*, in Proceedings of the 13th international conference on World Wide Web, WWW '04, New York, NY, USA, 2004, ACM, pp. 553–562.
[30] Paul McFedries, *The cloud is the computer. IEEE Spectrum Online, : http://www.spectrum.ieee.org/aug08/6490*, 2008.
[31] A. Pokahr, L. Braubach, and W. Lamersdorf, *Jadex: A bdi reasoning engine*, in Multi-Agent Programming, J. D. R. Bordini, M. Dastani and A. E. F. Seghrouchni, eds., Springer Science+Business Media Inc., USA, 9 2005, pp. 149–174. Book chapter.
[32] R. Akkiraju, J. Farrell, J.Miller, M. Nagarajan, M. Schmidt, A. Sheth, K. Verma, *Web Service Semantics WSDL-S. A joint UGA-IBM Technical Note, version 1.0: http://lsdis.cs.uga.edu/projects/METEOR-S/WSDL*, 2005.
[33] D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel, *Wsmo - web service modeling ontology*, in DERI Working Draft 14, vol. 1, BG Amsterdam, 2005, Digital Enterprise Research Institute (DERI), IOS Press, pp. 77–106.
[34] Roy Bragg, *Cloud computing: When computers really rule: http://www.technewsworld.com/story/63954.html*, 2008.
[35] K. S. Trivedi and R. Sahner, *Sharpe at the age of twenty two*, SIGMETRICS Perform. Eval. Rev., 36 (2009), pp. 52–57.
[36] S. Venticinque, R. Aversa, B. D. Martino, and D. Petcu, *Agent based cloud provisioning and management: design and protoypal implementation*, in Proc. of Cloud Computing and Services Science (CLOSER), SciTePress, 2011, pp. 184–191.
[37] VV.AA., *Cloud Computing Interoperability Forum, Unified Cloud Computing: http://code.google.com/p/unifiedcloud/.*
[38] ———, *Cloud Computing Interoperability Forum,Cloud taxonomy : http://groups.google.com/group/cloudforum/web/ccif-cloud-taxonomy.*
[39] ———, *Open Cloud Manifesto, Spring 2009 : http://www.opencloudmanifesto.org.*
[40] ———, *Open Grid Forum: Open Cloud Computing Interface (OCCI): http://forge.ogf.org/sf/projects/occi-wg.*
[41] ———, *Papyrus uml: http://www.papyrusuml.org.*
[42] ———, *Web Service Modelling Ontology (WSMO): http://www.wsmo.org.*
[43] A. Weiss, *Computing in the clouds*, netWorker, 11 (2007), pp. 16–25.
[44] M. Wooldridge, *Agent-based software engineering*, in IEE Proceedings on Software Engineering, 1997, pp. 26–37.
[45] L. Youseff, M. Butrico, and D. D. Silva, *Towards a unified ontology of cloud computing*, in Grid Computing Environments Workshop, 2008. GCE '08, Nov 2008, pp. 1–10.
[46] Q. Zhang, L. Cheng, and R. Boutaba, *Cloud computing: state-of-the-art and research challenges*, J. Internet Serv Appl, (2010), pp. 7–18.
[47] Y. Zhang, E. Manisterski, S. Kraus, V. Subrahmanian, and D. Peleg, *Computing the fault tolerance of multi-agent deployment*, Artificial Intelligence, 173 (2009), pp. 437 – 465.

Edited by: Dana Petcu and Daniela Zaharie
Received: March 1, 2012
Accepted: April 15, 2012

Scalable Computing: Practice and Experience
Volume 13, Number 1, pp. 45–57. http://www.scpe.org

ISSN 1895-1767
© 2012 SCPE

AN INTERNET OF THINGS PLATFORM FOR REAL-WORLD AND DIGITAL OBJECTS[*]

SUPARNA DE, TAREK ELSALEH, PAYAM BARNAGHI, AND STEFAN MEISSNER [†]

Abstract. The vision of the Internet of Things (IoT) relies on the provisioning of real-world services, which are provided by smart objects that are directly related to the physical world. A structured, machine-processible approach to provision such real-world services is needed to make heterogeneous physical objects accessible on a large scale and to integrate them with the digital world. The incorporation of observation and measurement data obtained from the physical objects with the Web data, using information processing and knowledge engineering methods, enables the construction of "intelligent and interconnected things". The current research mostly focuses on the communication and networking aspects between the devices that are used for sensing amd measurement of the real world objects. There is, however, relatively less effort concentrated on creating dynamic infrastructures to support integration of the data into the Web and provide unified access to such data on service and application levels. This paper presents a semantic modelling and linked data approach to create an information framework for IoT. The paper describes a platform to publish instances of the IoT related resources and entities and to link them to existing resources on the Web. The developed platform supports publication of extensible and interoperable descriptions in the form of linked data.

Key words: Interent of Things, Modelling, Linked-data, Semantics, Information model

1. Introduction.

The Internet of Things (IoT) vision aims to enable the machine perception of the real world and seamless interactions with it. This vision is lent credence with the growing availability of smart objects that are directly related to the physical world and have the communication and computation capabilities to connect and interact with their surrounding environment. The data and/or services offered by such objects can provide information about the physical world and allow interaction with it. The services can either be information services exposing functionalities that can provide data on the surrounding physical world entities, or actuation services that can bring about a change in the state of the physical world objects. Initially, the IoT vision considered physical objects tagged with RFID transponders. However, this has grown to encompass sensor networks and distributed smart objects collaborating via local networks or through the Internet [9]. Thus, the resulting real-world data/services need to be defined and made available in a homogeneous way to allow integration of the data from the wide variety of heterogeneous sources and to support autonomous reasoning and decision making mechanisms. This points to the applicability of Semantic Web technologies that can provide a formal, structured and machine-processible platform to heterogeneous data sources, as well as providing context to the data and to the objects themselves. Initial efforts in this area have resulted in ontologies for sensor descriptions [6] as well as standardisation efforts towards semantic descriptions of sensor networks [17]. However, the semantic sensor descriptions need to be linked to the measurements and domain knowledge and then to the observed IoT entity in the domain.

Another key requirement for the IoT is a platform that facilitates the virtualisation of real world objects. Existing research works have focused on sensor (and actuator) middleware frameworks that offer sensor descriptions [1], sensor site data [10] and measurement data services [16] on the Web and/or at the application level. More generic approaches include those that provision flash interfaces for instantiating semantic profiles of connected objects [7]. To extend this to heterogeneous real world objects, the data from the physical world needs to be interlinked to domain knowledge and existing data sources on the Web. This can facilitate automated annotation and reasoning on the physical world data and lead to provisioning of intelligent applications.

Towards these twin aims, this paper describes the Sense2Web platform that allows publishing data descriptions for the components of the IoT domain in the form of Linked Data and makes this data available to other Web applications via SPARQL endpoints. The platform offers both a Human-to-Machine (H2M) and Machine-to-Machine (M2M) interface for publishing data. It incorporates a semantic description framework for the IoT components and provides a formal representation to the interactions. The platform allows generating of networked resources which collect data from the physical world as well as data and services on the Web. Interlinking data from the physical world and the Web supports the provision of networked knowledge [11].

[*]This work is supported by the EU IoT-A project, IoT-A: Internet of Things Architecture (http://www.iota. eu/public) contract number: 257521. The second and third authors are also supported by the EU PPP FI-Ware project (http://www.fi-ware.eu/).

[†]Centre for Communication Systems Research, University of Surrey, Guildford, GU2 7XH, Surrey, UK ({S.De, P.Barnaghi, T.Elsaleh, S.Meissner}@surrey.ac.uk).

This also enables other related data and relevant information to be discovered and facilitates interconnection and integration of data from different communities and sources. The applicability of the platform is illustrated through a reference application scenario that implements a mash-up application using the generated linked data.

The rest of the paper is organised as follows: section 2 presents challenges and background information on linked data. The proposed information models are detailed in section 3. Section 4 presents the developed Sense2Web platform architecture and explains linking IoT concept descriptions to existing data sources on the Web. An example application that builds upon the platform functionalities is presented in section 5. Section 6 concludes the paper and discusses future work.

2. Challenges and background information.

In this section we first discuss the issues related to annotation and publication of the IoT related data and making the data machine-interpretable to support automated scenarios. We then provide the principle of creation and publication of the linked data.

2.1. Challenges.

In order to use the Linked data approach for publishing data from heterogeneous IoT concepts, we need to address the following challenges:

1. How to annotate "'plain"' data to make it semantically linked data: this refers to deciding what ontologies need to be leveraged to semantically describe the IoT domain. Moreover, the heterogeneity of possible IoT concepts requires using several ontologies together and this in turn, gives rise to the challenge of aligning and relating them to each other. This paper proposes a suite of ontologies to define an IoT information model. The ontologies build upon existing vocabularies and where appropriate, properties are included to allow linking the proposed ontologies to external ontologies where the given concept may be more completely described.
2. How to actually "'link"' the data together: the Linked Data principle is to "'make data refer to each other so that it eventually forms a data network"' [22]. However, currently, most of the data linkages are made manually or are very sparse [22]. The Human-to-Machine interface of the proposed platform offers automated hints for linking entered data to existing internal and external data repositories.
3. How to serve the published data in an application-programmable compliant way: currently, sensor network data applications apply Device Profile for Web Services (DPWS) [15] -based implementations [1], [14] to sensor gateways to offer sensor measurement data services. DPWS defines a limited set of WS-* standards for resource limited devices. The majority of Linked data is currently served through SPARQL endpoints.

The Sense2Web platform offers the published IoT component descriptions as Linked data through SPARQL endpoints.

2.2. Background Information - Linked data.

Publishing data on the Semantic Web with machine interpretable representations facilitates more structured and efficient access to the resources; however semantic descriptions without being linked to other existing data on the Web would be mostly processed locally and according to the domain descriptions (i.e. domain ontologies). Linking data to other resources enables obtaining more information related to a particular data item by exploring the links across different concepts and domains. The linked data concept was initially introduced by Tim Berners-Lee in 2006 [4]. Berners-Lee suggested four main principles to publish linked data: - using URIs as names for data, - providing HTTP access to those URIs, - providing useful information for URIs using the standards such as RDF and SPARQL, - Including links to other URIs. Publishing annotated and interlinked data is the underlying principal of creating linked Web resources that is referred to as the Web of Data [4]. In the Web of Data resources are connected via links that can be queried and interpreted using discovery and search agents [5]. Linked data enables navigation between different data sources by following the data connection links. This allows the linked data consumers to start with one data source and then browse through a vast number of resources interconnected by machine interpretable links (e.g. RDF links).

3. IoT Information Models.

This section defines the main abstractions and concepts that underlie the IoT domain and describes the relationships between them. The main tenet of the IoT is extension of the Internet into the physical world, to involve interaction with a physical entity in the ambient environment. The entity constitutes 'things' in the Internet of Things and could be a human, animal, car, store or logistic chain item, electronic appliance or a closed or open environment. The 'entity' is the main focus of interactions by humans and/or software agents. This interaction is made possible by a hardware component, a 'device', which either attaches to an entity or is part of the environment of an entity so it can monitor it. The device allows the entity to be part of the digital world by mediating the interactions. The actual software component that provides information on the entity or enables controlling of the device, is a 'resource'. As implementations of resources can be highly dependent on the underlying hardware of the device, a 'service' provides a well-defined and standardised interface, offering all necessary functionalities for interacting with entities and related processes. The services expose the functionality of a device by accessing its hosted resources. Other services may invoke such low-level services for providing higher-level functionalities, for instance executing an activity of a specified business process. The relations between services and entities are modeled as associations. These associations could be static, e.g. in case the device is embedded into the entity; they could also be dynamic, e.g., if a device from the environment is monitoring a mobile entity. These identified concepts of the IoT domain and the relations between them are depicted in Figure 3.1.

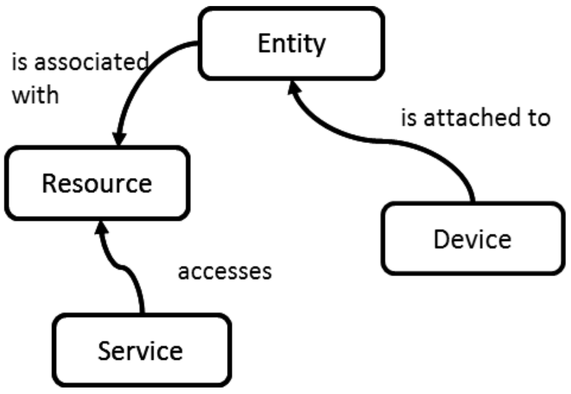


Fig. 3.1: IoT model: key concepts and interactions

Based on the identification above, of the main concepts in the IoT domain, this paper proposes a suite of ontologies that models entity, resources and IoT services. The ontologies are modelled in the Web Ontology Language - Description Logic (OWL-DL). Where appropriate, properties are included to allow linking the proposed ontologies to external ontologies; for example, the global location URI of an entity could link to the relevant location instance in the GeoNames ontology[1], where the given location is more fully described. This enables reusability of ontologies and fosters modularity.

3.1. Entity Model. In addition to the required properties of an identifier and some attributes, an entity can have certain other aspects that need to be taken into account. For example, we may need to know about the location of an entity and the features that can be observed by a sensing mechanism to provide data about the observed feature. A diagram of the main attributes of the entity model is shown in Figure 3.2.

An entity has certain features, which include domain attributes, temporal features and location (Entity:hasA U(DomainAttribute, TemporalFeatures, Location)). The OWL union operation (U) on these features denotes that a particular entity instance can have either or all of these features. Moreover, an entity instance can have multiple values for the domain, temporal or location feature.

Domain attributes tie the entity instance to a particular domain and a semantic realisation of the model can link the entity instance to a domain ontology. The domain attribute is specified in terms of the attribute name (*hasAttributeName*), attribute type (*hasAttributeType*) and value. These attribute properties together

[1]http://www.geonames.org/ontology/documentation.html

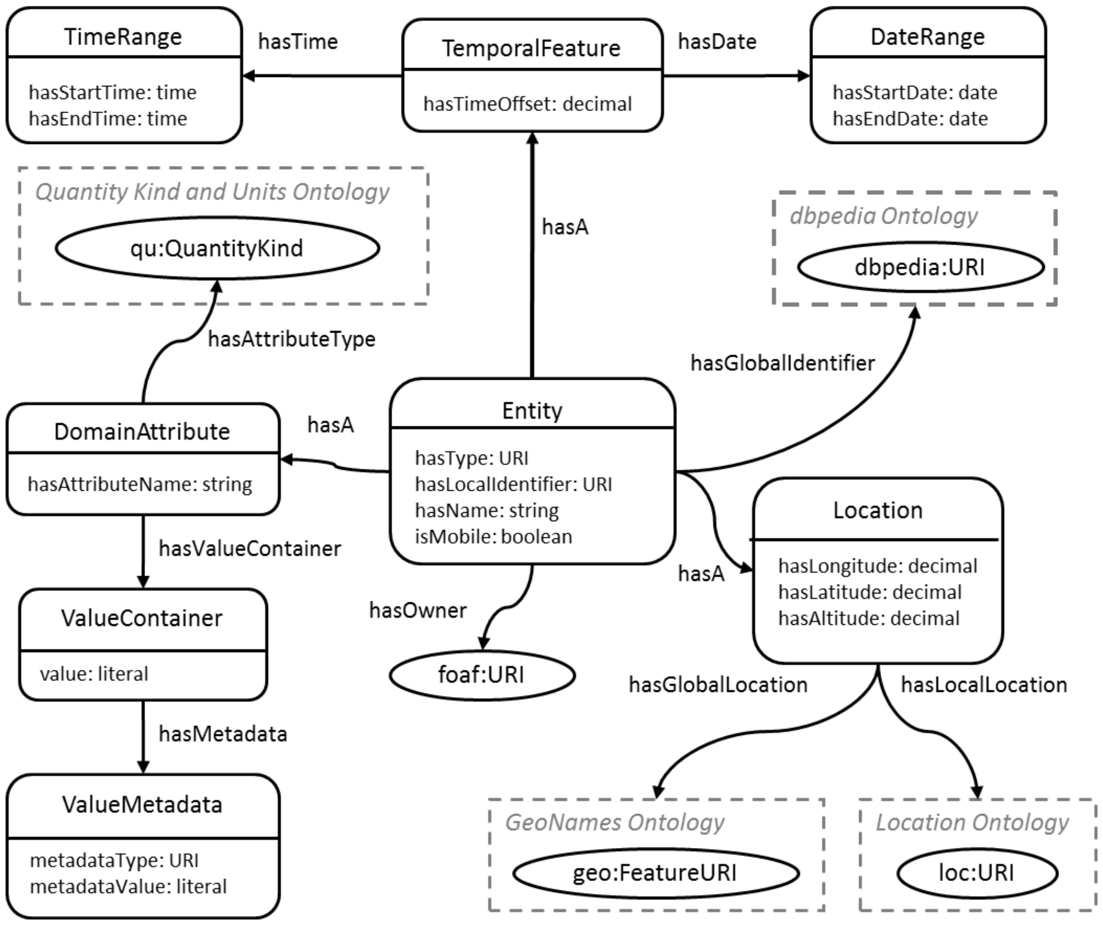


Fig. 3.2: The Entity model

describe an observable feature of the entity. Having the attribute name and type as distinct properties allows for two levels of data specification. The *DomainAttribute* instance's name property refers to the domain specific attribute of the virtual entity, e.g. Ambient Temperature. What a resource (e.g. sensor) will be able to measure will be the attribute type, i.e. Temperature, in this case. Thus, for two distinct domain attributes of the same virtual entity, e.g. Ambient Temperature and Body Temperature, what a resource would be concerned with, would be the attribute type, i.e. Temperature, which is the same for both domain attributes. Only the domain attribute property, which is intrinsic to the entity, puts what the resource senses, into context. The type of the "'*DomainAttribute*"' is further defined as "'*QuantityKind*"', taken from the "'Library for Quantity Kind and Units"' [13]. That library contains a list of physical phenomena, such as temperature or acceleration, which can be measured by sensors or influenced by actuators. The value itself has a literal 'value' and associated metadata information (*ValueMetadata*). The metadata could include information on, for instance, the units of measurement. It is specified in terms of the metadata value and metadata type.

The entity's lifetime is described by "'*TemporalFeature*""s further refined by "'*hastimeOffset*"', "'*TimeRange*"' and "'*DateRange*"'. The latter specifies intervals in a scale of days, months and years; "'*TimeRange*"' describes ranges in hours, minutes, seconds and fractions of seconds. A time offset to Coordinated Universal Time (UTC) in hours indicates the time zone the entity is currently located in. These capture the temporal properties of entities that may have temporal attributes, e.g. Meeting Rooms. The values of these properties can be compared with other dates by using date and time comparison built-ins (such as those available in Semantic Web Rule Language (SWRL) [23] to deduce facts about temporal aspects of the relevant entity.

Physical entities have a location at the time they exist in the real world. In this work, we focus on locations on the earth that can be described by geographic coordinates as well as symbolic locations, such as relative locations within a building. Barnaghi et al. [3] identify two location attributes for describing sensor data: the first attribute to refer to an instance of a local location ontology, which is a model of the current location offering high granularity and detailed information on the location in terms of the relative positioning of rooms, floors and buildings. The second location attribute was identified to be from a high-level concept available on the Web of data, such as DBpedia [2]. We adopt a similar approach in this paper and extend it to include specification of geographical coordinates for the entity location as well. Thus each 'Entity' can be given a "'*Location*"' that is modelled as a triple of float values describing longitude, latitude, and altitude as geographic position. The location concept also has properties that could link to local location (*hasLocation*) ontologies.

Additionally, an entity has datatype properties that specify the URI of an owner (*hasOwner*) where the URI could point to a foaf[2] profile, a literal name (*hasName*) and a Boolean property to denote if the entity could be mobile (*isMobile*). An important attribute of an entity is the entity type (*hasType*), which could be specified through the rdf:type property and hence, allow a Semantic Web engine to infer the type of the entity from its asserted properties, especially in cases where the entity could have multiple types. The local identifier (*hasLocalIdentifier*) property is the ID of the virtual entity. It could as well point to a local naming schema. The global identifier (*hasGlobalIdentifier*) property is a placeholder to associate the entity to the open Linked Data[3] platform; for instance, to a Dbpedia entry.

3.2. Resource Model. A resource is the core software component that represents an entity in the digital world. It allows the entity to be part of the digital world by mediating the interactions. Figure 3.3 details the resource description model.

The resource concept has *datatype* properties that specify its name (*hasName*), an ID (*hasResourceID*) and time offset (*hasTimeOffset*). The resource provider can specify certain keywords (or free text tags) describing the resource through the *hasTag* property. This is an optional property to allow the resource provider to provide a free text search for the resource instance. A resource also has a location property (*hasResourceLocation*) that links to the Location concept. This location could be the location of the device the resource runs on. The definition of the location concept is similar to that in the entity model. The resource type is denoted in terms of the type property (*hasType*) to the *ResourceType* concept. Resources can be instances of either of the following types: sensor, actuator, RFID tag, storage or processing resource. The different resource types are not disjoint, hence, resources can be an aggregation of several of these types. When the type is a sensor, the *hasType* property serves as a link to an instance of a sensor that conforms to an available sensor ontology (e.g. SSN sensor ontology). This allows linking the resource concept to external ontologies which already define in detail related concepts, without the need of repeating them in the resource model. Actuator resources modify the physical state of a physical entity. The RFID tag type is a specialised kind of sensing resource. A storage resource stores information obtained from other resources (such as sensors) and a processing resource includes methods to process the information aggregated from other resources (e.g. an aggregate of a temperature value coming from a number of sensors). As the access to a resource is provided by an IoT service, this link to the service is denoted by the "'*isExposedThroughService*"' object property that links the resource model to an IoT Service instance of the service model. The resource model also captures the link to the hardware 'device' on which it hosted (*isHostedOn*), which may be further described in a Device ontology.

3.3. Service Model. Resources are accessed by services which provide functionality to gather information about entities they are associated with or manipulate physical properties of their associated entities. The Service Model contains information needed for discovering and looking up the service as well as information on how to invoke the service. The service model is shown in Figure 3.4.

The actual technology used to invoke the service is modelled through the hasServiceType parameter, which could take a value such as 'REST' for a RESTful web service. The link to the resource to which the service

[2]http://www.foaf-project.org/
[3]http://linkeddata.org

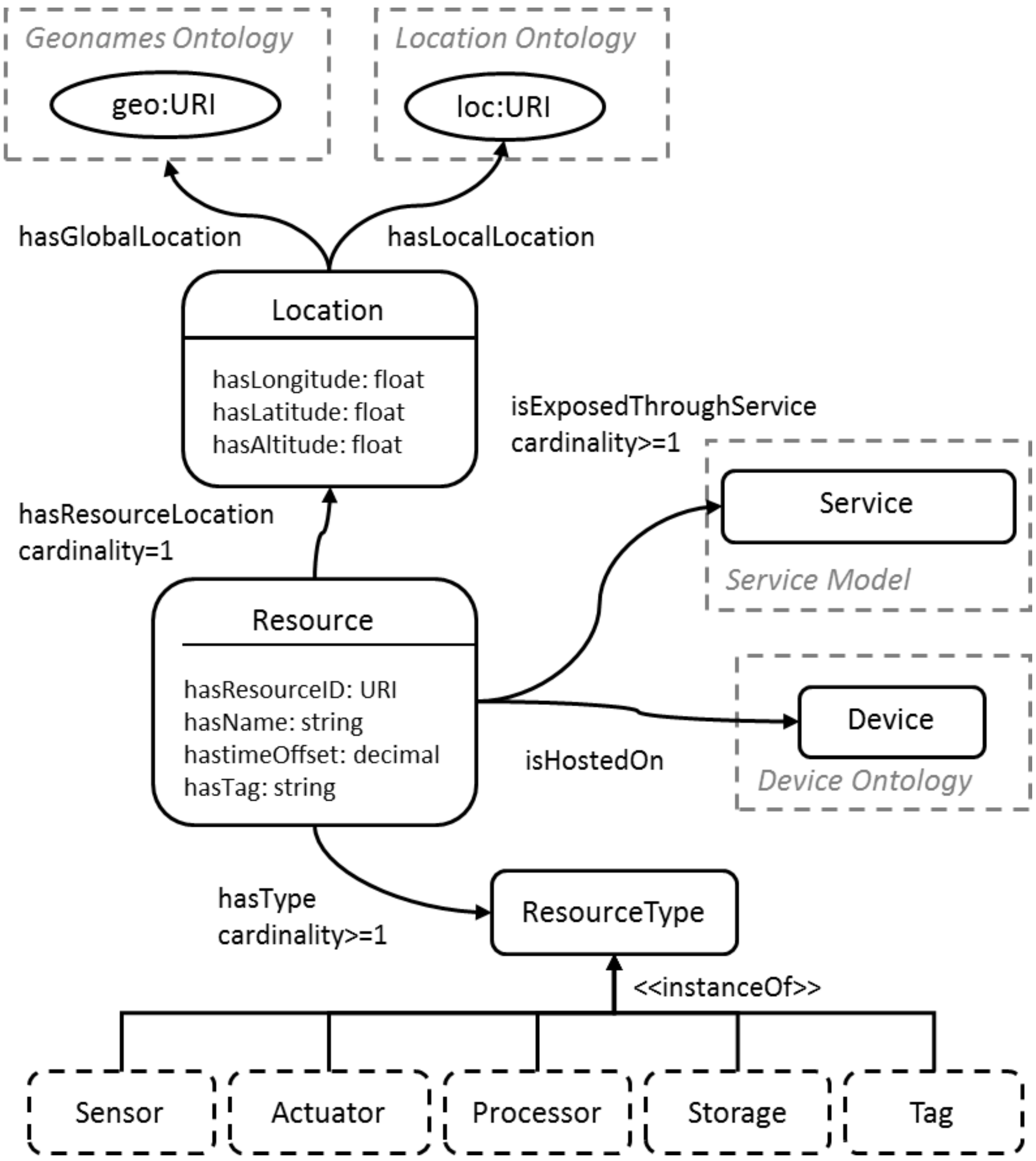


Fig. 3.3: The Resource model

provides access is specified through the exposes property that links back to an instance of the resource model.

One of the important aspects of a service is to allow for associations with virtual entities in the IoT domain. For this, the IoT-A proposed service model utilises the OWL-S [20] model as its upper ontology. The "'*ServiceModel*"' part of the OWL-S ontology is used to specify the input, output, preconditions and effects (IOPE) related parameters of the service model. Since the service model exposes the underlying resource's functionalities, the resource attribute that is exposed through an IoT service either as output data type (*hasOutput*) or as an input parameter (*hasInput*) is captured in the service specification. The feature can then be matched with the attribute type of the virtual entity with which it can be associated. For instance, a virtual entity can have an attribute that represents its "'*indoorTemperature*"'. The generic type of this particular attribute is "'*temperature*"'. Then, if there is a service exposed by a resource that measures temperature, specified as

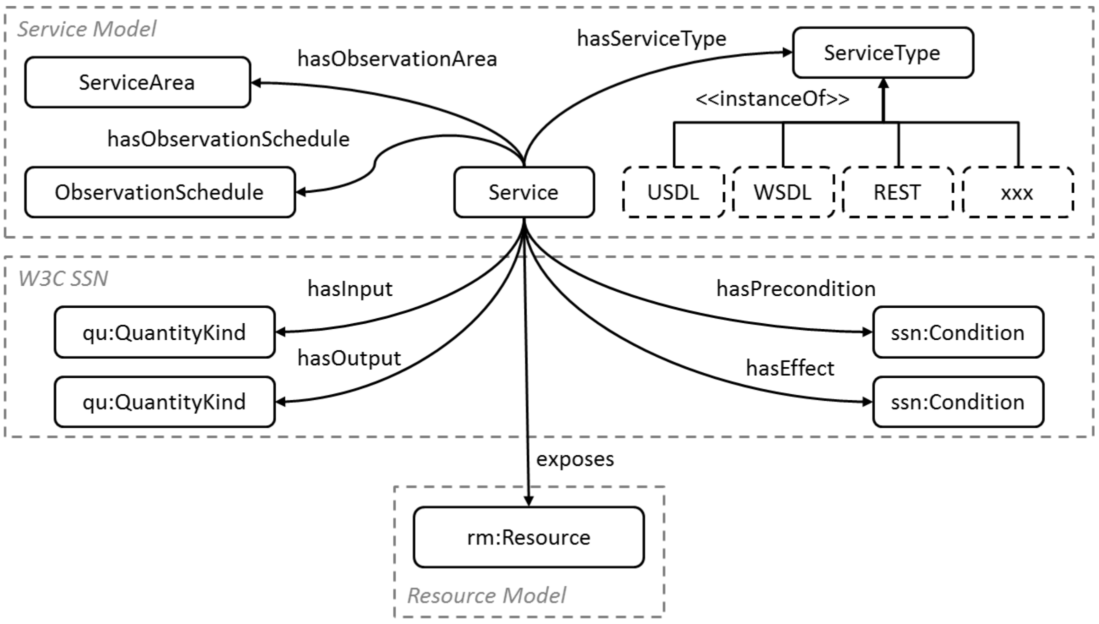


Fig. 3.4: The Service model

the service's *hasOutput* parameter, the corresponding service can be a candidate for possible association to the relevant virtual entity. The input and output parameters can be specified in terms of the generic instance quantities from the QU ontologies [18], such as "'*temperature*"' or "'*luminosity*"'.

For actuating services, the state of the entity attribute being controlled is also important. This post-condition state is modelled through the *hasEffect* parameter in the service model. Similarly, any pre-conditions that need to be met before the service execution can be specified through the *hasPrecondition* parameter. The state object properties link to instances of the 'Condition' class of the SSN ontology [17], so that conditions that affect the resource's measurement or actuating capabilities can be specified. This is also an example where the SSN ontology concepts can be extended to include actuating conditions.

With location being an important criterion for service search and resolution, the area affected by the service is specified through the *hasServiceArea* property. For sensing services, this would be the observed area, while actuating services would specify the area of operation. The observation area of sensors can be different to their actual location. An example for that are camera resources observing areas at some distance to their position. The possibility of specifying time constraints on service availability is captured through the *hasServiceSchedule* property. This can allow IoT users to be informed about downtimes of resources, for instance, for energy efficiency reasons.

4. Sense2Web Linked Data Platform. The Sense2Web platform[4], depicted in Figure 4.1, is a six-tiered framework for publishing linked IoT concept instances. The platform was developed in Java and deployed on the Apache Tomcat[5] web server.

The platform offers both a H2M as well as M2M interface for publishing IoT data and associating it to existing vocabularies on the Web. The core functions that are supported by the platform are essentially the CRUD (Create, Read, Update, and Delete) methods used for interacting with IoT entities and resources; in this case this translates to publishing, reading, updating and deleting the IoT concept descriptions. For all these

[4]An online version is available at: http://ccsriottb3.ee.surrey.ac.uk:8080/IOTA/
[5]http://www.tomcat.apache.org

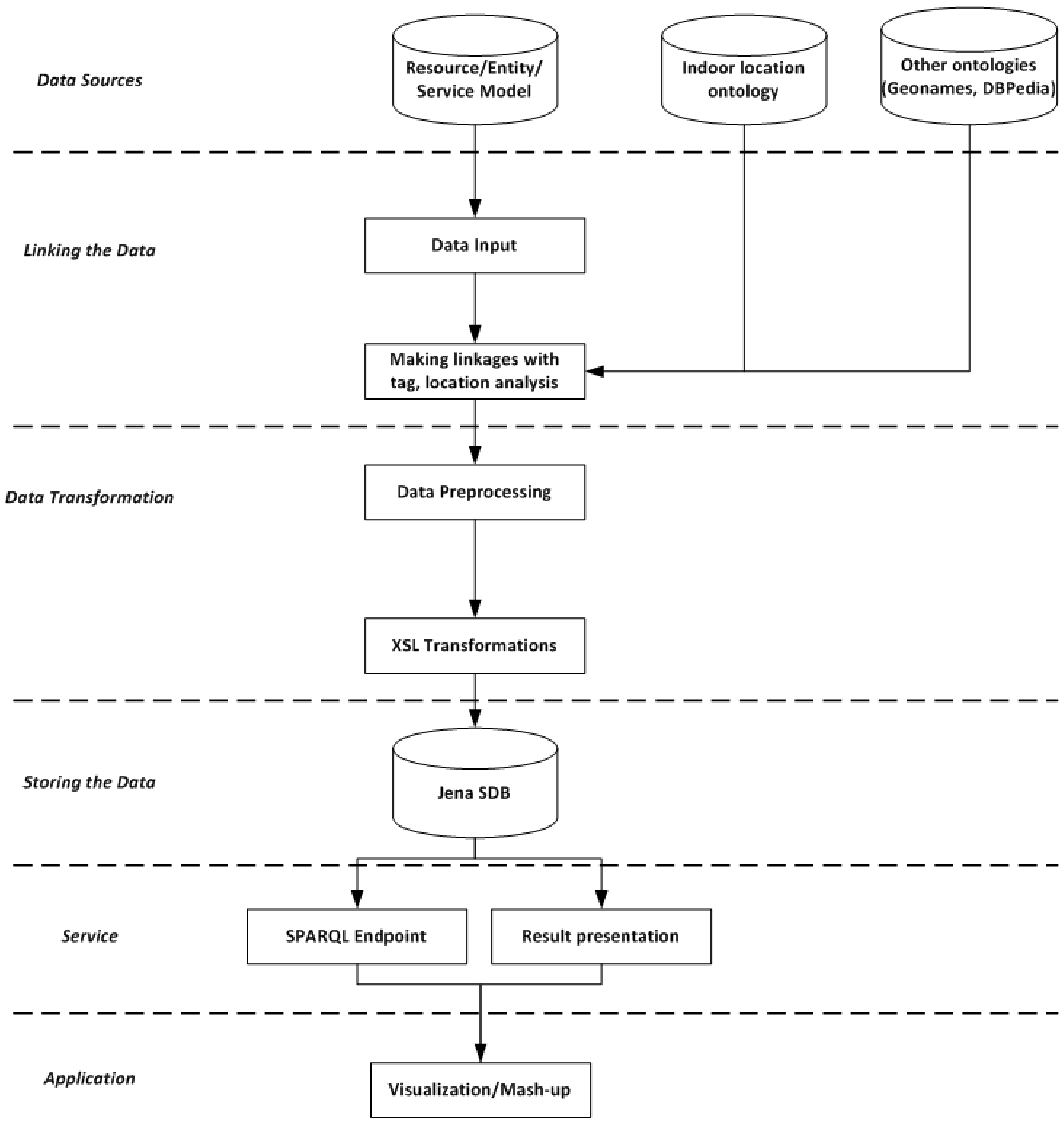


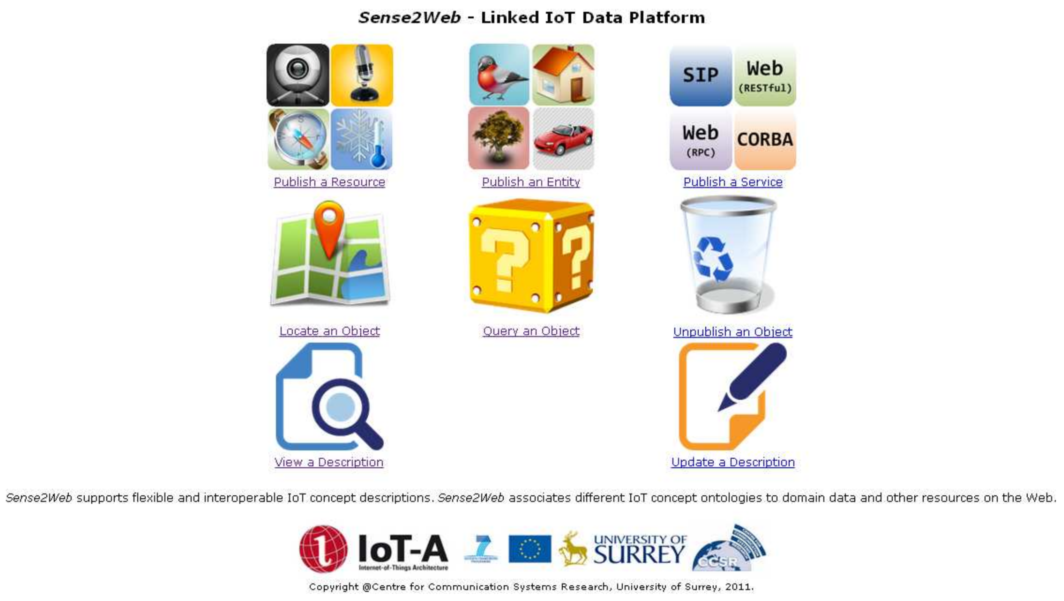
Fig. 4.1: Sense2Web architecture

methods, a web user interface is provided for H2M interaction, and RESTful interfaces are exposed for M2M interactions. The Sense2Web Web user interface is shown in Figure 4.2.

The different layers of the framework are explained below:

1. Data sources: the IoT information models detailing entities, resources and services that have been proposed in this paper (section 3) form the primary source of data structures for the H2M and M2M interfaces. In addition to these, the platform also accesses DBPedia and an indoor location ontology[6]

[6]http://ccsriottb3.ee.surrey.ac.uk:8080/IotaDataFiles/models/LocationModel.owl

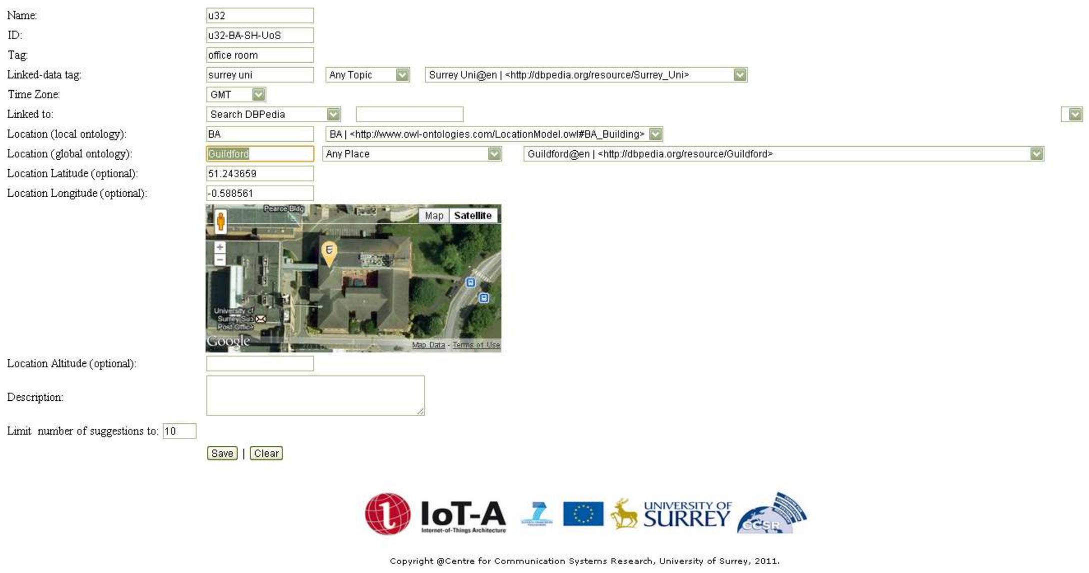


Sense2Web supports flexible and interoperable IoT concept descriptions. *Sense2Web* associates different IoT concept ontologies to domain data and other resources on the Web.

Copyright @Centre for Communication Systems Research, University of Surrey, 2011.

Fig. 4.2: Sense2Web user interface

to obtain values for location, type and descriptive properties.



Name:	u32		
ID:	u32-BA-SH-UoS		
Tag:	office room		
Linked-data tag:	surrey uni	Any Topic	Surrey Uni@en | <http://dbpedia.org/resource/Surrey_Uni>
Time Zone:	GMT		
Linked to:	Search DBPedia		
Location (local ontology):	BA	BA | <http://www.owl-ontologies.com/LocationModel.owl#BA_Building>	
Location (global ontology):	Guildford	Any Place	Guildford@en | <http://dbpedia.org/resource/Guildford>
Location Latitude (optional):	51.243659		
Location Longitude (optional):	-0.588561		
Location Altitude (optional):			
Description:			
Limit number of suggestions to:	10		

Copyright @Centre for Communication Systems Research, University of Surrey, 2011.

Fig. 4.3: Entity Publication H2M Interface

2. Linking the data: the H2M interface consists of a Web interface with a form to populate the elements of the object (entity/resource) description. This constitutes the data input stage; Figure 4.3 shows the H2M interface for publishing an entity description.
   To establish linkages with existing data repositories, we use Jena API to query the DBPedia and other resources and serialise the results using AJAX technology directly to the page; so the user can type a keyword and obtain relevant suggestions. For instance, in the Linked-data tag field, suggestions for relevant RDF links (URIs) to an input entered by the user in this field are retrieved from the DBPedia

knowledge base. This facilitates the interconnection and integration of data from different communities and sources. RDF link suggestions are also provided in the global and local location (from the indoor location ontology) fields with respect to the user's input. The form also provides location fields with respect to latitude and longitude. These fields make use of a Google mini-map which contains a marker which can be displaced to the position required, which will then populate the field with the respective co-ordinates.
The M2M interface is realised through a RESTful interface, developed using the Restlet API[7] that offers a RDF file upload option. The M2M interface is utilised for publishing instances of the service model.

3. Data transformation: When the form is submitted, the servlet handling the form processes the input data by collecting the fields and their respective values into an XML serialisation. This pre-processing step is succeeded by an Extensible Stylesheet Language Transformation (XSLT) [19] step which converts the input into an RDF instance that adheres to the corresponding IoT model (i.e. entity model in this case). This makes generation of the RDF data flexible and less dependent on the current model.
4. Storing the data: The RDF instance is then handed over to the SDB [12] interface, which then stores the RDF instance in SQL as nodes, triples, prefixes and quads.
5. Services: the platform supports retrieving, updating, or deleting a description, which can be done by providing the ID value of the published IoT concept. In addition to these methods, a SPARQL interface is provided for users to query for objects (resources or entities) or services of interest. Different results format are supported as well. The SPARQL [21] query page is shown in Figure 4.4.
6. Applications: an application can consume the services provided by the platform to make use of the published linked IoT concept instances. In the following section, we showcase a mash-up application that demonstrates the linked data usage and integration of data from different sources.

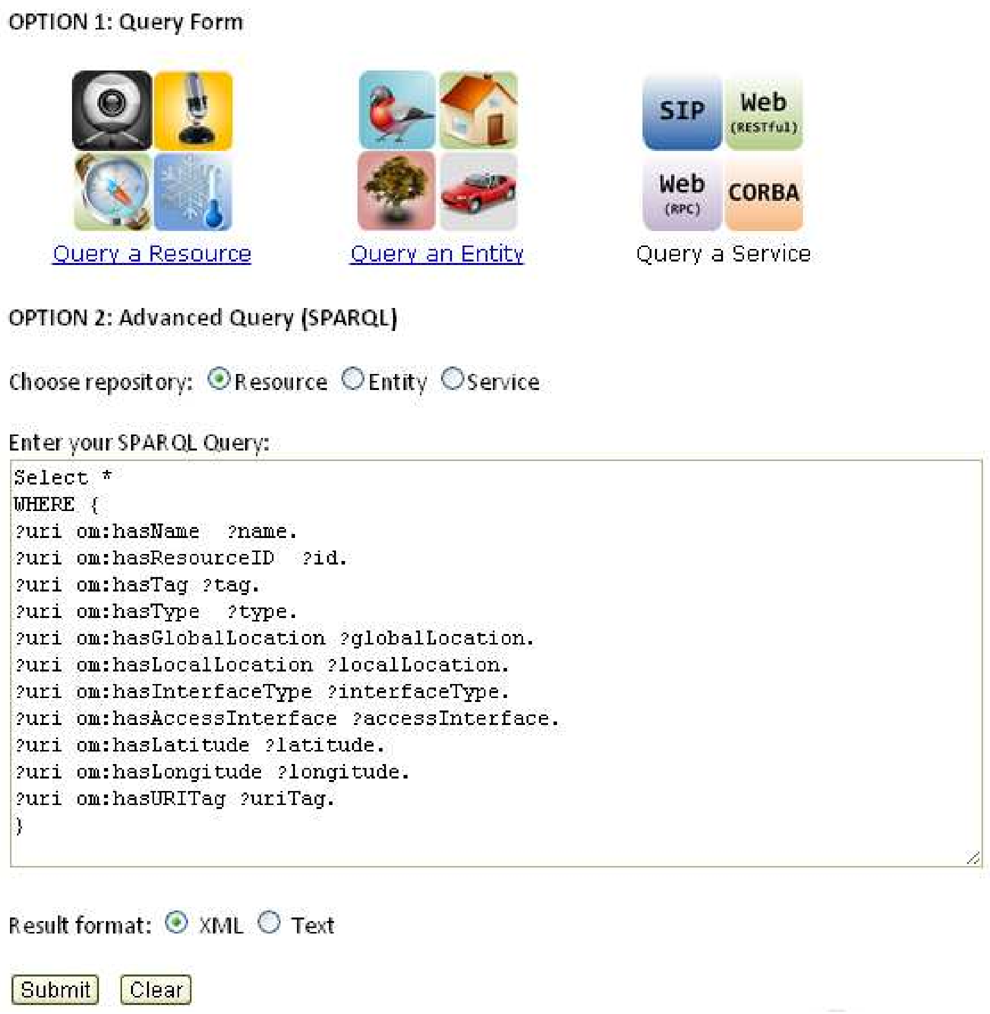
5. Google Maps Mash-up Application. The developed application is a map application that has been implemented using Google Maps API[8] to illustrate the location of the IoT instances and provide a summary of its description.

For this application, we use the location attributes and retrieve geographical coordinates of the resources and entities by processing the Linked data descriptions. The application retrieves related properties of the published IoT concept from the repository and lists available resources and entities through a Google Maps overlay. Figure 5.1 shows a screen-shot of the application and shows a published temperature sensing resource.

The map page refreshes periodically to show any changes in location that can be observed when object descriptions are updated. This is best noticed for example when remote sensor device gateways update Resource description when Resources migrate from gateway to gateway. The work in [8] has been integrated with the platform to demonstrate this scenario. In this scenario, a mobile sensor device attaches to a gateway. The gateway then creates a web service instance to expose the sensor resource to the web. The gateway also retrieves essential metadata from the sensor device and populates it in a RDF instance description which is then published to the Sense2Web platform via the M2M RESTful interface. As the sensor migrates to another gateway and re-attaches, the gateway will then update the description already stored at the platform, with the new location properties.

6. Discussion. To achieve scalability in real IoT deployments, the major issues involve providing semantic annotations, publishing the metadata, supporting large-scale distributed repositories and indexing and query support over the data. Manual resource annotation and tagging the data can hinder publishing large number of resources. Automating mechanisms are required to publish the resource and entity descriptions directly into the repositories. In a different work [24], we have studied and implemented a gateway component for large-scale sensor networks that publishes semantically annotated resource descriptions when the resources are discovered and associated to the gateway. This can help to automate the semantic annotation of resources. We have also implemented RESTful (M2M) interfaces that support direct publication and edit/update of the resources. The

[7] http://www.restlet.org
[8] http://code.google.com/apis/maps/



Choose repository: [✓ Resource] Entity Service

Enter your SPARQL Query:

```
Select *
WHERE {
?uri om:hasName  ?name.
?uri om:hasResourceID  ?id.
?uri om:hasTag ?tag.
?uri om:hasType  ?type.
?uri om:hasGlobalLocation ?globalLocation.
?uri om:hasLocalLocation ?localLocation.
?uri om:hasInterfaceType ?interfaceType.
?uri om:hasAccessInterface ?accessInterface.
?uri om:hasLatitude ?latitude.
?uri om:hasLongitude ?longitude.
?uri om:hasURITag ?uriTag.
}
```

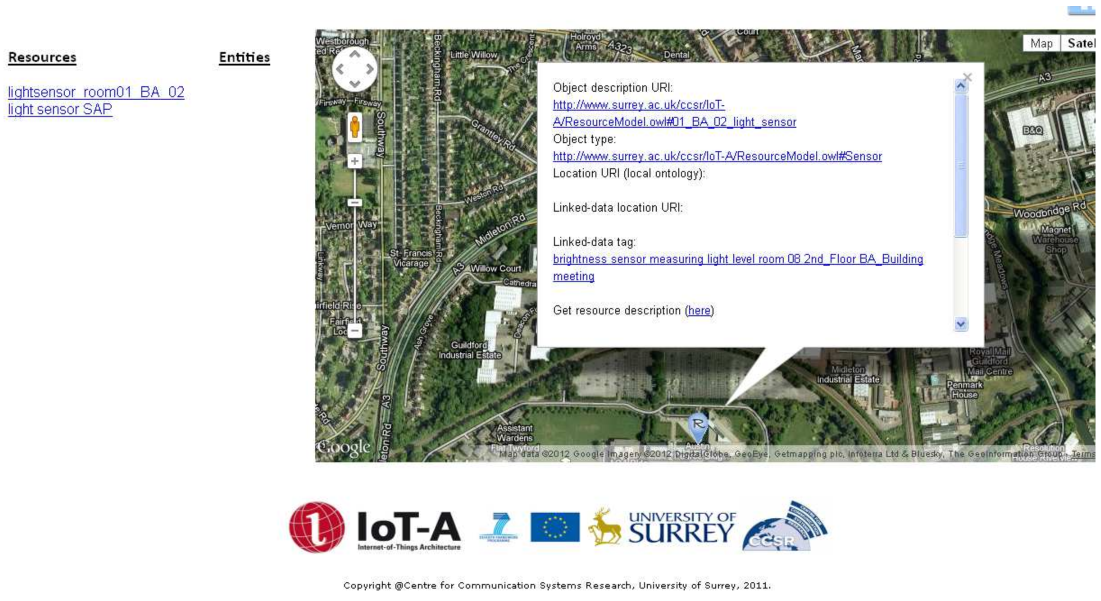
Result format: [✓ XML] Text

Fig. 4.4: SPARQL Query page

interfaces can be accessed directly by third-party applications and software agents and can support automated semantic annotation and query of the resources. Federation of repositories and coordinating search and query over a number of semantic data stores in multiple domains can be also supported by publishing data in different domain repositories. The domain repositories can be defined based on network domain, geographical distribution or other aspects that can help to distribute the data more efficiently and then queries can be distributed based on the selected features. Peer-to-Peer communication and data update in the repositories is also another issue to enable up-to-date and efficient distribution and publishing of semantic annotation. These aspects will be investigated in future extensions of the presented platform.

7. Conclusions and Future Work. This paper presents a set of interlinked semantic models for the IoT domain and describes a platform that provisions both a H2M and M2M interface for publishing data descriptions conforming to the developed semantic models in the form of Linked Data. The platform allows making this data available to other Web applications via standard SPARQL endpoints. The models proposed in this paper are designed based on our previous work and experiences in the SENSEI project[9] and SSN ontology. The proposed models provide associations between different components in the IoT domain. The models support a

[9]http://www.sensei-project.eu/



Resources **Entities**

lightsensor_room01_BA_02
light sensor SAP

Object description URI:
http://www.surrey.ac.uk/ccsr/IoT-A/ResourceModel.owl#01_BA_02_light_sensor
Object type:
http://www.surrey.ac.uk/ccsr/IoT-A/ResourceModel.owl#Sensor
Location URI (local ontology):

Linked-data location URI:

Linked-data tag:
brightness sensor measuring light level room 08 2nd_Floor BA_Building meeting

Get resource description (here)

Map
Satel
Google
Map data ©2012 Google Imagery ©2012 DigitalGlobe, GeoEye, Getmapping plc, Infoterra Ltd & Bluesky, The GeoInformation Group - Terms
Copyright @Centre for Communication Systems Research, University of Surrey, 2011.

Fig. 5.1: Google Maps mash-up application

semantic annotation framework so the legacy data can be also enhanced using these descriptions. The semantic annotation allows that the model data is represented as linked data and can be associated with the existing data on the Web and in particular Linked Open Data. Future work will involve development of a resolution framework that allows searching the large scale data of the instances of the models in the IoT domain and also automated inference of dynamic associations that can be identified by exploring and reasoning the interlinked descriptions.

REFERENCES

[1] Abangar, H., Barnaghi, P., Moessner, K., Tafazolli, R., Nnaemego, and A., Balaskandan, K., *A Service Oriented Middleware Architecture for Wireless Sensor Networks*, In Proceedings of Future Network & Mobile Summit 2010, Florence, Italy.

[2] Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, C., and Ives, Z., *DBpedia: A Nucleus for a Web of Open Data*, In Proceedings of the 6th International Semantic Web Conference (ISWC2007), 2007.

[3] Barnaghi, P., Presser, M., and Moessner, K,*Publishing Linked Sensor Data*, In Proc. 3rd International Workshop on Semantic Sensor Networks (SSN), in conjunction with the 9th International Semantic Web Conference (ISWC 2010), 2010.

[4] Berners-Lee, T., *Linked data*, Retrieved from http://www.w3.org/DesignIssues/LinkedData.html

[5] Bizer, C., Heath, T., Idehen, K., and Berners-Lee, T., *Linked data on the web (ldow2008)*, In WWW '08: Proceeding of the 17th international conference on World Wide Web, New York, NY, USA. (pp. 1265-1266), 2008.

[6] Bowers, S., Madin, J. S., and Schildhauer, M. P., *A Conceptual Modeling Framework for Expressing Observational Data Semantics*, In Q. Li et al. (Eds.), ER 2008, LNCS 5231, pp. 41-54, 2008.

[7] Christophe, B., Verdot, V., and Toubiana, V., *Searching the "'Web of Things"'*, In Proc. Fifth IEEE International Conference on Semantic Computing, Palo Alto, CA. (pp. 308 - 315), 2011.

[8] Elsaleh, T., Gluhak, A., and Moessner, K., *Service Continuity for Subscribers of the Mobile Real World Internet*, In Proc. IEEE International Conference on Communications Workshops, 2011.

[9] Gluhak, A., Krco, S., Nati, M., Pfisterer, D., Mitton, N., and Razafindralambo, T., *A Survey on Facilities for Experimental Internet of Things Research*, IEEE Communications Magazine, 49(11), 58 - 67, 2011.

[10] Guinard, D., Trifa, V., Karnouskos, S., Spiess, P., and Savio, D., *Interacting with the SOA-Based Internet of Things: Discovery, Query, Selection, and On-Demand Provisioning of Web Services*, IEEE Transactions on Services Computing, 3(3), 223-235.

[11] Hauswirth, M., and Decker, S., *Semantic reality - connecting the real and the virtual world*, In Microsoft SemGrail Workshop. (pp. 21-22), 2007.

[12] Jena, SDB - persistent triple stores using relational databases. Retrieved from http://incubator.apache.org/jena/documentation/sdb/index.html

[13] Koning, H. P. d., Rouquette, N., Burkhart, R., Espinoza, H., and Lefort, L., *Library for Quan-*

tity Kinds and Units: schema, based on QUDV model OMG SysML(TM), 2009, Version 1.2. doi: http://www.w3.org/2005/Incubator/ssn/ssnx/qu/qu.html

[14] Leguay, J., Lopez-Ramos, M., Jean-Marie, K., and Conan, V., *Service oriented architecture for heterogeneous and dynamic sensor networks*, In Proceedings of the Second international Conference on Distributed Event-Based Systems, 2008.

[15] OASIS. (2009). Devices Profile for Web Services (DPWS). Retrieved from http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01

[16] Spiess, P., Karnouskos, S., Guinard, D., Savio, D., Baecker, O., Souza, L., and Trifa, V., *SOA-based integration of the Internet of things in enterprise services*, In Proceedings of IEEE ICWS, Los Angeles, CA, USA, 2009.

[17] W3C SSN Incubator Group Report. Retrieved from http://www.w3.org/2005/Incubator/ssn/wiki/Incubator_Report

[18] SySML, O. Library for Quantity Kinds and Units: schema, based on QUDV model. 1.2. Retrieved from doi:http://www.w3.org/2005/Incubator/ssn/ssnx/qu/qu

[19] XSL Transformations (XSLT). W3C Recommendation v 1.0. from http://www.w3.org/TR/xslt

[20] OWL-S: Semantic Markup for Web Services. Retrieved from http://www.w3.org/Submission/OWL-S/

[21] SPARQL Query Language for RDF. W3C Recommendation, from http://www.w3.org/TR/rdf-sparql-query/

[22] Yu, L., and Liu, Y., *Using the Linked Data Approach in a Heterogeneous Sensor Web: Challenges, Experiments and Lessons Learned*, In Proc. Sensor Web Enablement (SWE) Workshop, Banff, Alberta, Canada, 2011.

[23] Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., and Dean, M., *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*, W3C Member Submission, 2004. Retrieved from http://www.w3.org/Submission/SWRL/

[24] Ganz, F., Barnaghi, P., Carrez, F., and Moessner, K., *Context-aware Management for Sensor Networks*, In Proceedings of the 5th International Conference on COMmunication System softWAre and middlewaRE (COMSWARE11), 2011.

Edited by: Dana Petcu and Daniela Zaharie
Received: March 1, 2012
Accepted: April 15, 2012

Scalable Computing: Practice and Experience
Volume 13, Number 1, pp. 59–71. http://www.scpe.org

ISSN 1895-1767
© 2012 SCPE

THE EFFECT OF TEMPORARY LINKS IN RANDOMLY GENERATED NETWORKS OF CONSTRAINTS

IONEL MUSCALAGIU[*], HORIA EMIL POPA [†] AND NEGRU VIOREL[‡]

Abstract. Additional communication links between unconnected agents are used in asynchronous searching, in order to detect obsolete information. A first way to remove obsolete information is to add new communication links, which allow a nogood owner to determine whether this nogood is obsolete or not. The second solution consists in temporarily keeping the links. A new link is maintained until a fixed number of messages have been exchanged through it. This article investigates different values for the number of messages, values that are either statically or dynamically, during the run time, determined. In the case of processing all the messages, we adapt a dynamical solution for determining the number of necessary messages for maintaining a connection. The experiments show a better efficiency in comparison with the standard Asynchronous Backtracking. In this paper we examine the effect of temporary links for the random binary constraints problem. Experiments with asynchronous search techniques are conducted on randomly generated networks of constraints. Experimental results show that the dynamical solution for the temporary links allows obtaining better results for the majority of classes of problems investigated.

Key words: Distributed constraint programming, asynchronous searching techniques, multiagent systems, messages.

AMS subject classifications. 68T20, 68T42, 68W15

1. Introduction.

Constraint programming is a programming approach used to describe and solve large classes of problems such as searching, combinatorial and planning problems. Lately, the AI community has shown increasing interest in the distributed problems, which are solvable through modeling, done by constraints and agents. The idea of sharing various parts of the problem among agents that act independently and collaborate in order to find a solution by using messages has proved itself useful. It has also led to the formal problem known as the Distributed Constraint Satisfaction Problem (DisCSP) [12], [13], [6]. DisCSPs are composed of agents, each owning its local constraint network. Variables in different agents are connected by constraints. Agents must assign values to their variables so that all constraints between agents are satisfied.

There are complete asynchronous searching techniques for solving the DisCSP, such as the ABT (Asynchronous Backtracking), AWCS (Asynchronous Weak Commitment), ABTDO (Dynamic Ordering for Asynchronous Backtracking) and DisDB (Distributed Dynamic Backtracking) [2, 5, 12, 13, 15, 6]. Starting from the algorithm of Asynchronous Backtracking (ABT), there has recently been suggested, in [2] a unifying framework, a starting kernel for some of the asynchronous techniques. From this kernel, several techniques have been derived, known as the ABT family. They differ in the way they store nogoods, but they all use additional communication links between unconnected agents to detect obsolete information. These techniques start from a common core (called the ABT kernel) which can lead to some of the known techniques, including the algorithm of Asynchronous Backtracking, by means of eliminating the obsolete information among agents.

Several solutions for the elimination of the old information among agents were suggested in [2], such as adding temporary links. A first way to remove obsolete information is to add new communication links to allow a nogood owner to determine whether this nogood is obsolete or not. These added links were suggested in the original ABT algorithm.

A second solution (called by its authors ABTtemp, in [2]) consists in temporarily keeping those links between the agents that cannot determine if an information is outdated or not. This algorithm adds new links between the agents during the search, same as ABT. The difference is that new links are temporary. A new link is maintained until a fixed number of messages have been exchanged through it. After that, it is removed.

Different values for the number of messages are investigated in [7]. These values are either statically determined (before the run) or dynamically determined during runtime. A dynamical solution for determining the number of necessary messages for maintaining a connection is suggested in [7]. The first experiments show a better efficiency in comparison with the standard Yokoo version. The dynamic solution is based on determining the outdated nogood message flow and using that information for determining the number of messages.

Starting from the dynamical solution for determining the necessary number of messages needed for keeping a temporary link, in [8] is suggested a new hybrid method for eliminating the outdated information between

*Faculty of Engineering of Hunedoara, "Politehnica" University of Timisoara, Romania (ionel.muscalagiu@fih.upt.ro).
†Faculty of Mathematics and Informatics, West University of Timisoara, Romania (hpopa@info.uvt.ro)
‡Faculty of Mathematics and Informatics, West University of Timisoara, Romania (vnegru@info.uvt.ro).

the agents. This solution consists in transforming some of the temporary links into permanent links, based on the information about the outdated message flow. Applying this method to the ABT kernel, we can obtain a new hybrid technique, that takes what's best from the two derived techniques: ABT and ABT temporary link. A new dynamical solution for determining the number of necessary messages for maintaining a connection is suggested in this paper in the context of processing all the messages.

In a previous research presented in [8], the evaluation of the effect of temporary links is done using a particular problem: the problem of coloring a graph in the distributed versions.

The evaluation of the asynchronous search techniques depends on at least two factors: the types of problems used at the evaluation and the units of measurement used. There are a few types of problems about the evaluation in the DisCSP literature: the distributed problem of the m-coloring of a randomly generated graph and the randomly generated (binary) CSP. These problems are characterized by the 4-tuple (n,m,p1,p2), where: n is the number of variables; m is the uniform domain size; p1 is the portion of the n * (n - 1) /2 possible constraints in the constraint graph; p2 is the portion of the m*m value pairs in each constraint that are rejected by the constraint [11].

It must be mentioned that the randomly generated binary CSP are the most suitable for the evaluation, because they allow different densities for the constraints graph and they have many direct applications in real practice. Therefore, a complete evaluation supposes the selection of a varied class of problems - the more randomly chosen sets of data or the choice of sets of data which allow varied densities for the constraints graph. In this paper, extensive evaluation of the asynchronous search techniques with temporary links is conducted on randomly generated networks of constraints.

In a previous research [8], the evaluation of the effect of temporary links is done using NetLogo environment. NetLogo is a programmable modeling environment, which can be used for simulating certain natural and social phenomena [14]. Also, the NetLogo is a programming environment with agents that allows the implementation of the asynchronous techniques ([14], [16], [17]).

The evaluations from [8] were implemented using certain particularities, supplied by NetLogo, related to the asynchronous run of the agents. The agents work with the specific command "ask-concurrent". A command like this will allow launching the message treating routine, which is specific to each agent. Of course, each agent works asynchronously with the messages, but at the end of a command's execution there is a synchronization of agents' execution, synchronization that particularizes, in a way, the implementations being used. The evaluations performed in [8] are realized in particular conditions, which don't affect the generality of the results.

In order to make such estimation, in this paper these techniques are implemented in NetLogo. The implementation and evaluation is done using the extended model suggested in [9], model that is called DisCSP-NetLogo. Implementation examples for the ABT family can be found on the website [17]. In [9] a general implementation and evaluation model with synchronization and support for message management in Netlogo, for the asynchronous techniques is proposed. This model will allow the use of the NetLogo environment as a basic simulator for the study of asynchronous search techniques. This model can be used in the study of the agents' behavior in several situations, like the priority order of the agents, the behavior in the synchronous and asynchronous case.

2. The Framework.

This paragraph presents some notions related to the DisCSP modeling, ABT algorithm [12], [13], [6] and ABT family, [2].

2.1. The Distributed Constraint Satisfaction Problem.

The Distributed Constraint Satisfaction Problem (DisCSP) has been formalized in [12], [13].

Definition 2.1. *The model based on constraints CSP - Constraint Satisfaction Problem, existing for centralized architectures, is defined by a triple (*X, D, C*), where:* X=$\{x_1,...,x_n\}$ *is a set of n variables; whose values are taken from finite domains* D= $\{D_1, D_2,...,D_n\}$*;* C *is a set of constraints declaring those combinations of values which are acceptable for variables.*

The solution of a CSP implies to find an association of values for all the variables that satisfy all the constraints.

Definition 2.2. *A problem of satisfying the distributed constraints (DisCSP) is a CSP, in which the variables and constraints are distributed among autonomous agents that communicate by exchanging messages. Formally, DisCSP is defined by a 5-tuple (*X, D, C, A, ϕ*), where* X, D *and* C *are as before,* A $= \{A_1,...,A_p\}$ *is a set of* p *agents, and* ϕ : X $\longrightarrow$ A *is a function that maps each variable to its agent.*

In this article we will consider that each agent A_i has allocated a single variable x_i, thus $p = n$. Also, we assume the following communication model [12], [13]:

- agents communicate by sending messages. An agent can send messages to other agents iff the agent knows the addresses of the agents.
- the delay in delivering a message is finite, although random. For transmission between any pair of agents, messages are received in the order in which they were sent.

The Asynchronous Backtracking algorithm uses 3 types of messages:

- the *ok* message, which contains an assignment variable-value, is sent by an agent to the constraint-evaluating-agent in order to see if the value is right.
- the *nogood* message, which contains a list (called nogood) with the assignments wherefore a looseness was found, is sent in case the constraint-evaluating-agent finds an unfulfilled constraint.
- the *add-link* message, sent to announce the necessity to create a new direct link, caused by a nogood appearance.

Definition 2.3. *Two agents are connected if there is a constraint among the variables associated to them. Agent A_i has a higher priority than agent A_j if A_i appears before A_j in the total ordering. Agent A_i is the value-sending agent and agent A_j the constraint-evaluating agent.*

Definition 2.4. *The $agent - view$ list belonging to an agent A_i is the set of the newest associations received by the agent for the variables of the agents to whom it's connected.*

Definition 2.5. *The nogood list is a set of associations for distinct variables for which an inconsistency was found (an unsatisfied constraint).*

The $agent - view$ list together with the stored *nogood* values constitutes the working context of each agent, depending on them the agent makes decisions.

Definition 2.6. *A nogood list received by agent A_i is consistent for that agent, if it contains the same associations as $agent - view$ for all the variables of the parent agents A_k connected with A_i.*

Definition 2.7. *A nogood message is outdated if it contains a nogood list that isn't consistent with the receiver's agent context.*

ABT requires links to be directed. A constraint causes a directed link between the two constrained agents: the value-sending agent, whence the link departs, and the constraint-evaluating agent, to which the link arrives. When the value-sending agent makes an assignment, it informs the constraint-evaluating agent, which tries to find a consistent value. If it cannot, it sends back a message to the value-sending agent to cause backtracking. To make the network cycle free there is a total order among agents, which is followed by the directed links. In this article the lexicographical order is used.

Each agent keeps its own agent view and nogood store. Considering a generic agent, its own agent view is the set of values that are assigned to agents connected to it by incoming links. A nogood is a subset of agent view. If a nogood exists, it means the agent cannot find a value from the domain consistent with the nogood. When agent A_i finds its agent-view including a nogood, the values of the other agents must be changed. The nogood store keeps nogoods as justifications of inconsistent values. Agents exchange assignments and nogoods. When a random agent makes an assignment, it informs those agents connected to it by outgoing links. The agent always accepts new assignments, updating its agent-view accordingly. When it receives a nogood, it accepts it if the nogood is consistent with the agent's own agent view, otherwise it is discarded as obsolete (outdated nogood messages). An accepted nogood is added to the agent's nogood store to justify the deletion of the value it targets. When the agent cannot take any value consistent with its agent-view, because of the original constraints or because of the received nogoods, new nogoods are generated as inconsistent subsets of the agent-view, and are sent to the closest agent involved, causing backtracking. The process terminates when achieving quiescence, meaning that a solution has been found, or when the empty nogood is generated, meaning that the problem is unsolvable.

2.2. The ABT Family. Starting from the algorithm of Asynchronous Backtracking (ABT), in [2], several derived techniques were suggested, based on this one and known as the ABT family. They differ in the way that they store nogoods, but they all use additional communication links between unconnected agents to detect obsolete information. These techniques are based on a common core (called ABT kernel) hence some of the known techniques can be obtained, including the algorithm of Asynchronous Backtracking, by eliminating the old information among the agents. In [2] the starting point is a simple procedure that includes the main characteristics of the asynchronous search algorithms. Starting from this procedure, which forms the unifying framework, one can reach the known algorithms or variants that are close to them: Asynchronous Backtracking (ABT), Distributed Dynamic Backtracking (DisDB), Distributed Backtracking algorithm (DIBT) [2], [5], [12].

The ABT kernel algorithm requires, like ABT, that constraints are directed - from the value-sending agent to the constraint-evaluating agent - forming a directed acyclic graph. Agents are ordered statically in agreement with constraint orientation. Agent i has higher priority than agent j if i appears before j in the total ordering. In this article we will consider the lexicographical order for the agents, order used also in the case of the Asynchronous Backtracking algorithm. Considering a generic agent self, $\Gamma^{-}(self)$ is the set of agents constrained with self appearing above it in the ordering, also called the set of the parents of $self$. Conversely, $\Gamma^{+}(self)$ is the set of agents constrained with self appearing below it in the ordering, also called the set of the childrens of $self$.

The ABT kernel algorithm is a new ABT-based algorithm that does not require to add communication links between initially independent agents. The ABT kernel algorithm is sound but may not terminate (the ABT kernel may store obsolete information). In [2] were suggested several solutions for the elimination of the old information among agents, solutions that are summarized hereinafter.

A first approach to remove obsolete information is to add new communication links to allow a nogood owner to determine whether this nogood is obsolete or not. These added links were suggested in the original ABT algorithm.

A second way to remove obsolete information is to detect when a nogood could become obsolete. In that case, the hypothetically obsolete nogood and the values of unrelated agents are forgotten. These two alternative ways lead to the following four algorithms:

1. Adding links at preprocessing: ABT_{all}. This algorithm adds all the potentially useful new links during a preprocessing phase. New links are permanent.
2. Adding links during search: ABT. This algorithm adds new links between agents during search. A link is requested by self when it receives a Back message containing unrelated agents above self in the ordering. New links are permanent.
3. Adding temporary links: ABT_{temp}. This algorithm adds new links between agents during search, as ABT. The difference is that new links are temporary. A new link is maintained until a fixed number of messages have been exchanged through it.
4. No links: $DisDB$. No new links are added among the agents. To achieve completeness, this algorithm has to remove obsolete information in finite time. To do so, when an agent backtracks, it forgets all nogoods that hypothetically could become obsolete.

In [8], we proposed a new solution for combining the two methods for eliminating the outdated information, solution that will lead to the fifth hybrid algorithm:

5. Adding temporary (dynamic) links: ABT with permanent and temporary links (ABT_{TPL}). This new algorithm adds new links during the search. A part of these links are temporary, they are kept until a certain number of messages is exchanged (number determined dynamically during runtime). In exchange, some temporary links are transformed in permanent links, based on some information regarding the maximal flow of outdated nogood values.

3. Asynchronous Backtracking with temporary and permanent links. ABT with permanent and temporary links requests links dynamically, exactly like ABT. When a new link is set from agent i to j, it is maintained for a fixed number k of Info messages going from A_i to A_j. After this number of messages has been sent, the link is removed and agents A_i and A_j become disconnected. The number k of messages for a link is known a priori by both agents.

Some solutions for determining the number k of messages exchanged by the agents with temporary links are suggested and analyzed in [7]. The solutions presented are of two types:

1. statical solutions - for which the number of messages is fixed and doesn't change during the run time;
2. dynamical solutions - for which the number of messages varies during the run time.

The suggested statical solutions are based on determining a value for k common for all the agents, which is determined statically, at the beginning. The statical version supposes the construction of the induced graph associated to the problem (in a preprocessing phase). To each DisCSP problem we can associate a constraint graph, in which the nodes are agents/variables, and the edges are given by the existence of the constraints between agents/variables. From this constraint graph we can obtain the induced graph, corresponding to the existing order, by adding links between the parents of each node (the nodes from $\Gamma^{-}(self)$), if those links don't already exist. That graph is built as in [4]: agents (graph nodes) are processed from last to first, when an agent (graph node) is processed, all its parents (related agents before it in the ordering) are connected by new links if they were not connected before.

Based on this graph, we can determine a fixed number of messages k for all agents, as follows: the number of messages will be equal to the greatest value of the numbers of neighbors of each agent in the induced graph.

The dynamic versions proposed in [7] and [8] are based on using the information regarding the outdated nogood message flow. That information changes during the run time. As we know, when the agent receives a nogood, it is accepted if it is consistent with its own agent view, otherwise it is discarded as obsolete (outdated nogood messages). The outdated message flow also increases because the agents are not informed (because of the nonexistence of the supplementary links). Thus, each agent uses a supplementary data structure, for retaining the number of outdated nogood messages encountered at a given time. Those values are used for the determination of the number of messages exchanged for each temporary link. Practically, that value is the greatest number of nogood messages received at a given time.

Definition 3.1. *For each agent we have a local list of counter variables for counting the number of outdated messages received (named $COldNogood$). Let $MaxNrOldNogood$ be the maximum value from the $COldNogood$ list.*

So, we start with a fixed value for the number of messages, equal to the largest number of neighbors from the induced graph. This initial value is updated during the run time, using the largest value of the number of outdated messages, from all the agents.

The experiments presented in [7] and [8] show that the dynamical solution for determining the number of messages is the most efficient.

The solution suggested in [8] consists in transforming some temporary links in permanent links. In fact, the temporary links with those agents with which A_i has exchanged a maximal number of outdated nogood messages are transformed in permanent links. For each agent is determined the agent A_j with which A_i had exchanged the maximal number of outdated messages (item A_j *COldNogood* = *MaxNrOldNogood*), among those with which it had temporary links. The temporary link that exists with that agent will be transformed into a permanent link.

The agents exchange among themselves the values of $MaxNrOldNogood$ in order to determine and use the maximum one. That solution supposes that each agent knows the maximum number of outdated messages received by each agent ($MaxNrOldNogood$). A solution is based on the transmission of $MaxNrOldNogood$ of each agent to the ones it is connected with, in the moment of the transmission of an info or nogood message. The idea is presented in [8]. Each agent, in the moment of transmitting a message, attaches the value for the maximum flux of outdated messages, value stored in $MaxNrOldNogood$. In exchange, at the receiving of a message from an agent A_k that contains the maximum value of it, *SenderMaxOldNogood* will update the value of the $MaxNrOldNogood$.

In figure 3.1 we show those changes required in the ABT technique (version derived from the core ABT kernel), based on the method of determining temporary and permanent links. We obtain a new hybrid technique, technique that uses whats best from both of the derived techniques: ABT and ABT temporary link. The lines from the figure 3.1 marked with two digits are additional to the algorithm from [2] and the ones marked with *** contain modifications to those from the cited algorithm.

The obtaining of the version with temporary and permanent links supposed many changes in the basic ABT kernel algorithm.

First of all, each agent will use two extra sets $\Gamma_e^+(self)$ and $\Gamma_e^-(self)$, for the identification of the child and parent agents that appear because of the temporary links. In procedure $ABTkernel()$, in lines 1.1. and 1.2. they are determined. Also, it is necessary to introduce two data structures $CMessageTempLink$ and $COldNogood$. The first structure will be used by an agent A_i to retain the number of info messages transmitted for each temporary link. The second structure is used for counting the number of received outdated messages.

The new algorithm needed the introduction of a fourth message *RemoveL*, which notified a child agent about the canceling of a temporary link between two agents. Practically, the child agent will cancel the Sender agent from the list of its parents. The required changes appear in line 9.2. from procedure $ABTkernel()$ and procedure $RemoveLink()$, (a newly added routine).

Third after selecting a new value and announcing the child agents about the new selection, it is necessary that the verification of temporary links determines how many of them remain actual. This thing is done in procedure $CheckAgentView(msg)$ line 3.1, by calling a new procedure named $CheckRemoveLink()$. That routine verifies, for child agents from $\Gamma_e^+(self)$, if the maximum number of messages that are transmitted for that link has been reached.

```
procedure ABTkernel()
1    myValue ← empty; end ← false;
1.1  Set $\Gamma_e^+(self) \leftarrow \emptyset$ ***
1.2  Set $\Gamma_e^-(self) \leftarrow \emptyset$ ***
2    CheckAgentView();
3    while (not end) do
4        msg←getMsg();
5        switch(msg.type)
6            Info : ProcessInfo(msg);
7            Back : ResolveConflict(msg);
8            Stop : end ← true;
9.1          AddL : SetLink(msg);
9.2          RemoveL: RemoveLink(msg); ***
end

procedure CheckAgentView(msg)
1    if not consistent(myValue;myAgentView) then
2        myValue← ChooseValue();
3        if (myValue) then
             for each child∈ $\Gamma^+(self)$ do
                 sendMsg:Info(child;myValue);
3.1          CheckRemoveLink() ***
4        else Backtrack();
end

procedure ProcessInfo(msg)
1    Update(myAgentView; msg.Assig);
2    CheckAgentView();
end

procedure ResolveConflict(msg)
1    if Coherent(msg.Nogood;$\Gamma^-(self) \cup \{self\}$) then
2.1      CheckAddLink(msg)
3        add(msg:Nogood;myNogoodStore);
4        myValue ← empty; CheckAgentView();
5.1  else
         if Coherent(msg.Nogood; self) then
             SendMsg:Info(msg.sender; myValue);
5.2      Replace item Sender COldNogood with
             item Sender COldNogood + 1 ***
end

procedure SetLink(msg)
1    add(msg.sender;$\Gamma^+(self)$);
2    add(msg.sender;$\Gamma_e^+(self)$); ***
3    sendMsg:Info(msg.sender; myValue);
end

procedure CheckAddLink(msg)
1    for each (var ∈ lhs(msg.Nogood))
2        if not (var ∈ $\Gamma^-(self)$) then
3            sendMsg:AddL(var,self);
4            add(var;$\Gamma^-(self)$); add(var;$\Gamma_e^-(self)$); ***
6            Update(myAgentView; var ← varValue);
end
```

```
procedure RemoveLink(msg) ***
1    remove(msg.sender;$\Gamma^-(self)$);
2    remove(msg.sender;$\Gamma_e^-(self)$);
end

procedure CheckRemoveLink() ***
1    for each child ∈ $\Gamma_e^+(self)$
2        if (item child COldNogood = MaxNrOldNogood ) then
             replace item Child FlagList with 1;
3        if (item child CMessTemporaryLink ≥ MaxNrOldNogood
                 and item Child FlagList = 0 ) then
4            remove(child;$\Gamma^+(self)$);
5            remove(child;$\Gamma_e^+(self)$);
6            sendMsg:RemoveL(child,self);
7            Update(myAgentView; var child ← unknown);
end

procedure Backtrack()
1    newNogood←solve(myNogoodStore)
2    if (newNogood = empty) then
3        end ← true; sendMsg:Stop(system);
4    else
5        sendMsg:Back(newNogood, $x_j$);
         /*where $x_j$ has the lowest priority in V */
6        Update(myAgentView;rhs(newNogood)←unknown);
7        CheckAgentView();
end

function ChooseValue()
1    for each v∈D(self)not eliminated by myNogoodStore do
2        if consistent(v; myAgentView) then
             return (v);
3        else
             add($x_j = val_j$) self ≠ v;myNogoodStore);
         /*v is inconsistent with xj 's value */
4    return (empty);
end

procedure Update(myAgentView; newAssig)
1    add(newAssig;myAgentView);
2    for each ng ∈ myNogoodStore do
3        if not Coherent(lhs(ng);myAgentView) then
             remove(ng;myNogoodStore);
end

function Coherent(nogood; agents)
1    for each var ∈nogood ∪ agents do
2        if nogood[var] ≠ myAgentView[var] then
             return false;
3    return true;
end
```

Fig. 3.1: The ABT algorithm with temporary and permanent links.

4. Experimental results. The evaluation of the asynchronous search techniques depends on at least two factors: the types of problems used for the evaluation and the metrics of measurement used. There are a few types of problems used for evaluation in the DisCSP literature:

- the distributed problem of the m-coloring of a randomly generated graph, characterized by the number of nodes/agents, k=3 colors and the m-number of connections between the nodes/agents. Two types of problems are defined: sparse problems (having m=n x 2 connections) and dense problems (m=n x 2.7).
- The randomly generated (binary) CSPs are characterized by the 4-tuple (n,m,p1,p2), where: n is the number of variables; m is the uniform domain size; p1 is the portion of the n * (n - 1) /2 possible constraints in the constraint graph; p2 is the portion of the m*m value pairs in each constraint that are disallowed by the constraint. That is, p1 may be thought of as the density of the constraint graph, and p2 as the tightness of constraints.

4.1. The randomly generated DisCSP. A randomly generated DisCSP is an example of a homogeneous unstructured problem [11]. These problems have a number of variables with a fixed domain. Variables belonging to constraints are chosen randomly. Specifically, we implemented and generated in NetLogo both solvable and unsolvable randomly generated DisCSPs. These problems had one variable per agent so all constraints are between variables belonging to different agents (inter-agent constraints). Specifically, a tuple $< n, d, p1, p2 >$ was generated, where n is the number of variables, d is the domain size of all variables, p1 is the constraint density and p2 is the constraint tightness.

We implement in NetLogo a random instance generator in two steps [17]:
S1: We select with repetition $nr(C) = p_1\frac{n(n-1)}{2}$ random constraints. Each random constraint is formed by selecting without repetition 2 of n variables.
S2: For each constraint we uniformly select without repetition $nr(v) = p_2 \cdot d^2$ incompatible tuples of values, i.e. each constraint relation contains exactly $1 - p_2 \cdot d^2$ compatible tuples of values.

Implementation examples for the random instance generator can be found on the website [17].

We used binary constraints with the constraint density controlling how many constraints were generated and the constraint tightness determining the proportion of value combinations forbidden by each constraint. For example, a constraint density of 0.4 would generate 40% of the possible constraints in the problem (i.e. (n* (n-1)/2) * 0.4 where n is the number of variables) and a constraint tightness of 0.5 would prevent 50% of the possible value combinations of variables involved in a constraint from satisfying the constraint. Such uniform random constraints networks of n variables, k values in each domain, a constraints density of p1 and tightness p2, are commonly used in experimental evaluations of DisCSP algorithms [2], [3], [6].

The experiments were conducted on networks with 15-20 agents (n = 15 or n=20) and 10 values (k = 10). Three density parameters were used, p1 = 0.2, p1=0.4 and p1 = 0.5. In many cases a density of p1 = 0.2 or 0.3 was used to represent sparse constraint networks and a density of p1 = 0.4 or p1=0.5 used for medium networks. The value of p2 was varied between 0.3 to 0.5. This creates problems that cover a wide range of difficulty, from easy problem instances to instances that take several CPU minutes to solve. For every pair (p1,p2) in the experiments we present the average over 100 randomly generated instances (for each version we carried out a number of 100 trials, retaining the average of the measured values). Specifically, we tested the random classes: $< 20; 10; 0.2; 0.3 >$, $< 20; 10; 0.4; 0.7 >$, $< 20; 10; 0.2; 0.3 >$,$< 20; 10; 0.2; 0.5 >$, $< 20; 10; 0.5; 0.3 >$, $< 20; 10; 0.5; 0.5 >$ (100 solvable and unsolvable instances).

Another experiment is done for networks with n=15 agents, p1=0.4 (medium constraint networks) where the tightness value p2 varies between 0.1 and 0.9 to cover all ranges of problem difficulty. This aimed to test all algorithms near the phase transition region where some problem instances are very difficult to solve [6], [11].

4.2. Evaluation of temporary links for the ABT family. In order to make such estimation, the families of ABT techniques are implemented in NetLogo [14], [16], [17]. The implementation and evaluation is done using the two models proposed in [9].

In order to make the evaluation of the asynchronous search techniques, the message flow was counted i.e. the quantity of info (ok) and back (nogood) messages exchanged by the agents, the number of checked constraints i.e. the local effort made by each agent, and the number of nonconcurrent constraints checks (defined in [6], noted with ncccs) necessary for obtaining the solution.

Asynchronous techniques use some message processing routines. Those procedures process sequentially or in packages the messages that are in the message queues. Typically, each agent extracts one or more messages from its communication channel and calls the appropriate message processing routine. In this paper we analyze two classes of implementations:

- A version in which the messages are read and processed sequentially, one by one [2] -noted with ABT$_1$. In this version, we eliminate the redundant and outdated messages of the info type;
- A version of the ABT family with complete processing of messages: each agent treats entirely the existing messages in its message queue- noted with ABT$_2$.

We will present in this paragraph a protocol for message management for the ABT technique [3], [10] in the context of temporary links. This protocol establishes the order in which the messages are treated and the moment in which is tried the association of a new value. Also, this protocol allows complete or partial processing of the messages, by means of the use of the *msize* parameter, which stands for the number of messages read at a given time from the message queue. The *msize* parameter can take values between 1 and the length of the message queue. In the case that *msize* is 1, the sequential message processing solution is obtained.

The protocol presented here supposes the following:

P1. It is processed message by message:

- if it is of the info type, the local work context is updated (agent-view).
- the local counter *MaxNrOldNogood* is updated with *SenderMaxOldNogood* (received from another agent).
- if the message is of the back type, it's stored and verified if it is outdated. If it is outdated, an ok message is returned to the sender to inform him of that. A part of the back messages are thus rejected.
- if the message is of the addlink or removelink type then it's treated normally

P2. The current agent value is saved.
P3. The work context is updated, updating the nogood values.
P4. The routine check-agent-view is called.
P5. The neighboring agents are notified if the agent has kept its old value.

Starting from this protocol we propose a message management routine. This version is presented in fig. 4.1. As we can see in fig. 4.1 each agent can process all the messages until the message queue is emptied, or exactly as many messages as there are in the moment of the call, operation accomplished with the lines 1 and 1'.

The behaviors of several asynchronous techniques are investigated in two cases: the agents execute asynchronously the processing of received messages (the real situation from practice) and the synchronous case where the agents' execution is synchronized.

Seven implementations are done corresponding to the version presented:

- Variants Yokoo based on the asynchronous model from [9]: ABT-Y$_1$ (one message), ABT-Y$_2$ (complete processing of messages).
- Versions that determine statically the number of messages (named ABT-S$_1$ and ABT-S$_2$, corresponding to the static solutions presented in the previous paragraph).
- Versions that determine dynamically the number of messages: ABT-TPL$_1$ (one message), ABT-TPL$_{21}$ (complete processing messages - the solution proposed in this article) and ABT-TPL$_{22}$ (complete processing message - solution proposed in [3]) . These versions are corresponding to the ABT algorithm with temporary and permanent links presented in the previous paragraph.

Results appear in table 4.1, where we report the number of checked constraints (Constr.) the number of nonconcurrent constraint checks (Ncccs) and the total number of messages exchanged(Tmess), averaged over 100 executions.

In figures 4.2 and 4.3 are presented the results of other experiments for n=15 agents and p1=0.4 (medium constraint networks) where the tightness value p2 varies between 0.1 and 0.9 to cover all ranges of problem difficulty. This aimed to test all algorithms near the phase transition region where some problem instances are very difficult to solve [6], [11]. Figure 4.2 shows the computational effort, the number of nonconcurrent constrain checks, for all three versions of ABT. Figure 4.3 presents the total number of messages sent by the algorithms in the same run.

As known, the quantity of constraints checked evaluates the local effort done by each agent, but the number of nonconcurrent constraint checks count computational effort of concurrently running agents only once during each concurrent running instances citemeis1. Analyzing the results from table 4.1, one can notice that the

```
to message-manage [msize]
  set nrm 0
1 while [not empty? message-queue and nrm < msize] or
1' while [not empty? message-queue] ***
  [
     set msg retrieve-message
     if (first msg = "stop")
        [ stop ]
     if (first msg = "info")
        [ Update MyContext with msg
        [ Update $MaxNrOldNogood$ with $SenderMaxOldNogood$ ]    // if max COldNogood < SenderMaxOldNogood
     if (first msg = "back")
        [ Update $MaxNrOldNogood$ with $SenderMaXOldNogood$ ]    // if max COldNogood < SenderMaxOldNogood
        [ ifelse (Not Is-obsolete msgNogood Sender)
          [ Store msg to BackSet] //builds the list containing the received back messages
          [ SendInfo msg]
        // if it is outdated the sender agent is announced according to the Is-Obsolete procedure
        ]
     if (first msg = "addl")
        [ SetLink msg ]
     if (first msg = "removel")
        [ RemoveLink msg ]
     set nrm nrm + 1
   ]
  UpdateContextInfo
  Check-agent-view
  If Not empty(BackSet)
    [ ProcessMessageBackSet]
end
```

Fig. 4.1: The message-manage procedure for the message management in the case of the techniques from the ABT family

Table 4.1: The results for ABT2 versions (n=20)

n = 20 agents		p1= 0.2		p1= 0.5	
		p2=0.3	p2=0.5	p2=0.3	p2=0.5
ABT-Y_2	TMess	28	238100	89288	279378
	Constr.	1353	4434588	2275457	3495841
	Ncccs	499	1324866	357750	405249
ABT-S_2	TMess	78	570448	110475	273908
	Constr.	1380	14813543	3274523	3876091
	Ncccs	504	3883867	421890	438485
ABT-TPL_{21}	TMess	76	229272	79199	253178
	Constr.	1373	4420864	1982961	3270209
	Ncccs	500	1313842	322808	392978
ABT-TPL_{22}	TMess	71	278654	91054	214715
	Constr.	1422	5920563	2582961	4363385
	Ncccs	533	1454423	362718	491903

dynamical solution of $ABTTL$ reduces the local effort made by the agents. In case of problems with low density, the two solutions require approximatively the same costs (messages and global effort). An explanation is given by the fact that no temporary links appear, the only differences are caused by the delays in supplying the messages. The more the difficulty of the problems and the density of the constraint graph grow (p2=0.5 or p1=0.5), the more the costs of the dynamical solutions decrease. But, as the difficulty of the problems increases (n=20 agents, p2=0.5), the static solution $ABTS_2$ required much greater efforts compared to the dynamical variant ABT-TPL_{21}.

In the case of the message flow, the solutions with temporary links require a smaller flow of messages. Unfortunately, with the increase of the number of agents and the difficulty of the problems (p2=0,5) the static solutions for the temporary links require a much greater flow of messages. This thing is caused also because the temporary links aren't kept long enough to detect obsolete information.

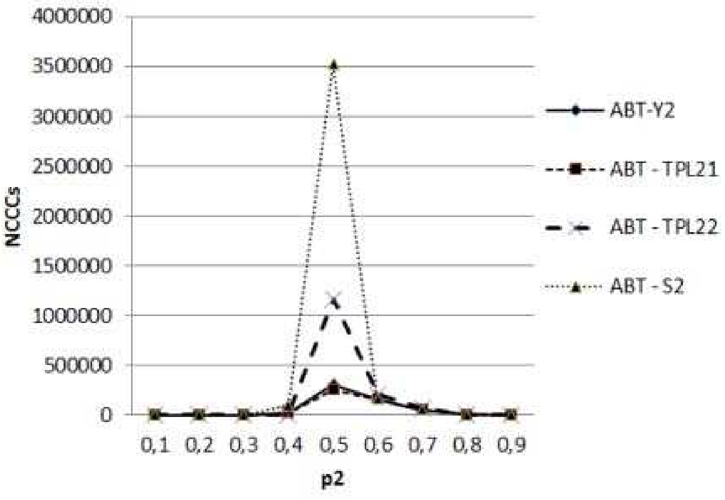


Fig. 4.2: The number of nonconcurrent constraint check for the ABT techniques.

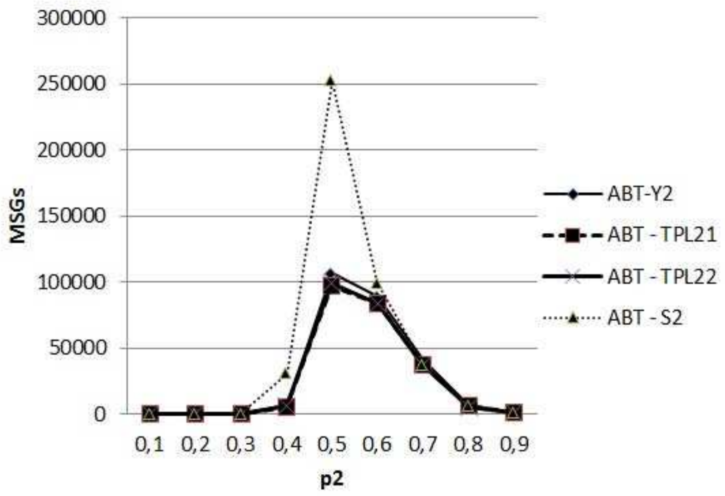


Fig. 4.3: Total number of messages sent for the ABT technique

As regarding the two dynamic solutions that use two different methods of treating the message in packages, the variant proposed here surpasses the one proposed in [3]. There can be observed situations in which the dynamical solution $ABTS_{22}$ is surpassed by the Yokoo solution.

Regarding the effort done by the agents,for the harder problem instances, ABT-TPL$_{21}$ outperforms ABTY by a factor of 1.1. Unfortunately, for the difficult problems we can observe a network load for the all solutions.

A version in which the messages are read and processed sequentially, one by one [2] - noted with ABT$_1$ is evaluated. This solution supposes a message treatment routine, which extracts sequentially each message, identifies its type and calls the appropriate processing routines. In this routine, for message processing, we eliminate the redundant and outdated messages of the info type.

In the case of the versions in which the messages are read and processed sequentially, one by one (noted with ABT$_1$) the results appear in table 4.2. These variants behaved similarly.

The results in the synchronous case where the agents' execution is synchronized appears in table 4.3 In other words, the agents perform a computing cycle in which they process a message from a message queue in the synchronous case. After that, a synchronization is done waiting for the other agents to finalize the processing

Table 4.2: The results for ABT1 versions - one message (n=20)

n = 20 agents		p1= 0.2	p1= 0.4
		p2=0.3	p2=0.7
ABT-Y_1	TMess	62	3290
	Constr.	2303	145422
	Ncccs	761	16362
ABT-S_1	TMess	63	3392
	Constr.	2330	153645
	Ncccs	766	18049
ABT-TPL_1	TMess	60	3220
	Constr.	2209	143001
	Ncccs	732	16174

of their messages. For this case we also count the number of cycles necessary obtaining the solution (Ncycles), which is a measure that could approximate the global effort (similar to NCCCs).

Table 4.3: The results for ABT1 versions - one messages (the synchronous case)

n = 20 agents		p1= 0.2	p1= 0.4
		p2=0.4	p2=0.7
ABT-Y_1	TMess	55	2474
	Constr.	2318	94958
	Ncccs	633	24176
	Ncycles	14	390
ABT-S_1	TMess	58	2079
	Constr.	2082	85439
	Ncccs	701	22746
	Nrcycles	14	370
ABT-TPL_1	TMess	55	2258
	Constr.	1949	89542
	Ncccs	622	23312
	Nrcycles	13	380

In this case, also we notice that the dynamical solution requires a lower flow of messages and also a lower global effort.

A general remark is that the static solutions applied to easy problems (low density or p2<0.4) require similar costs or even lower than all the other solutions. This thing is caused by the fact that the management of temporary links determines an extra overhead.

Unfortunately, analyzing the sets of results for certain instances (during runtime) we remarked the existence of problems for which the versions with temporary links (static versions) require very high costs. Although, we should specify that the number of those cases was not very high, not influencing, in the end, the results.

4.3. Discussion. It is interesting to see how many such links can be added by ABT during the search for a solution. The actual number will obviously depend on the instance to be solved, in [2] an estimate of the worst case is made, as follows: When a wipe out occurs on an agent A_i, the agent i builds a nogood by resolution of it's nogood store, and sends the obtained nogood to the agent A_j with the lowest priority in this set. When agent A_j receives the nogood, it checks the compatibility of the nogood with its own agent view. But, since this nogood can contain variables (x_k), unknown for agent A_j, agent A_j will ask the agents A_k to add a link from k to j. In the worst-case, a wipe out occurring at agent A_i will generate a nogood involving the whole set $\Gamma^-(i)$ of the agents linked to i, and preceding i in the agent ordering (the parents of node). More generally, when traveling back to all the ascendent agents, a nogood can lead to the addition of links between each pair of agents in $\Gamma^-(i)$, leading to a total number of links equal to $|\Gamma^-(i)|\ (|\Gamma^-(i)| + 1)\ /2$, see [2] for details.

The estimate presented previously was done in [2] for the worst case. In order to see, though, for the chosen data sets, how many links appear, during the experiments was counted also the number of temporary links. In figure 4.4 is presented the number of links for the chosen types of problems. An average was performed for all the runs and classes of problems. Surprisingly, this average is far from the values of the worst case. For problems in which p2 has small values (the constraint tightness) the number of temporary links is small, but for large values of p2≥4 the number of temporary links is almost the same.

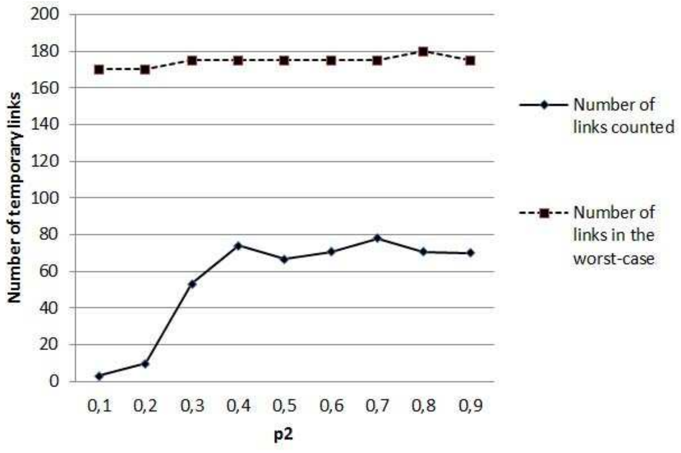


Fig. 4.4: The number of temporary links for ABT2 versions.

5. Conclusions. In this paper we examined the effect of temporary links for the random binary constraints problem. Experiments with asynchronous search techniques are standardly conducted on randomly generated networks of constraints. Experimental results illustrate that the dynamical solution for the temporary links has a better efficiency in comparison with the Asynchronous Backtracking.

A new dynamical solution for determining the number of messages necessary for maintaining a connection is proposed in this paper, the experiments show a better efficiency in comparison with the standard Asynchronous Backtracking.

The new member presumes transforming some of the temporary links in permanent links, based on information relative to the outdated message flux received by each agent.

From the experimental analysis we conclude that statical solutions proposed are not fitted for the case of networks with many links because they require a greater message flux. On the other hand, we remark a smaller general computing effort compared with the classical solution from [2], [12]. In conclusion it is recommended the use of dynamical variants that use message management and the agents work asynchronously.

A last comparison between the cases of processing the messages sequential or in packages, we can notice a neat differentiation between the dynamic solution and the classic or static solutions. The processing of all messages allows the agent to receive much faster the maximums of the other agents, compared to the situation in which it treats one message.

The scale-free graphs in complex networks, recently introduced by Barabasi and Albert [1], have become a very popular interdisciplinary research topic. As a future research, we wish to analyze temporary links in scale-free graphs, since there was little research in network structure for DisCSP.

REFERENCES

[1] A. L. Barabasi and A. L Albert, *Emergence of scaling in random networks*, Science, 286 (1999), pp. 509-512.

[2] C. Bessiere, I. Brito, A. Maestre and P. Meseguer,*Asynchronous Backtracking without Adding Links: A New Member in the ABT Family*, Artificial Intelligence, 161:7-24, 2005.

[3] I. Brito, P. Meseguer, *Synchronous, asynchronous and hybrid algorithm for DisCsp.* In Workshop on Distributed Constraints Reasoning, Toronto, 2004.

[4] R. Dechter and J. Pearl, *Network-based heuristics for constraint-satisfaction problems.* Artificial Intelligence, 34(1998), pp. 1–38.

[5] Y. Hamadi, C. Bessiere and J. Quinqueton, *Backtracking in distributed constraint networks.* In Proceedings ECAI'98, Brighton, UK, 1998, pp. 219–223.

[6] A. Meisels, *Distributed Search by Constrained Agents: algorithms, performance, communication*, Springer Verlag, London, 2008, pp. 105–120.

[7] I. Muscalagiu, H.E. Popa and M. Panoiu, *Determining the number of messages transmitted for the temporary links in the case of ABT Family Techniques.* Proceedings of the 7th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania. IEEE Computer Society Press, 2005.

[8] I. Muscalagiu, H.E. Popa and M. Panoiu, *Asynchronous Backtracking with temporary and fixed links: A New Hybrid Member in the ABT Family.* Journal of Computer Science INFOCOMP, Brazil, Vol. 5, nr. 2 (2006), pp. 29–37.

[9] I. Muscalagiu, H. Jiang, H.E. Popa, *Implementation and evaluation model for the asynchronous techniques: from a synchronously distributed system to a asynchronous distributed system.* Proceedings of the 8th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, IEEE Computer Society Press, 2006, pp. 209–216.

[10] H.E. Popa, I. Muscalagiu, D.M. Muscalagiu and V. Negru, *Experimental analysis of the impact of the message management in the case of the ABT family.* Proceedings of the 9th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania. IEEE Computer Society Press, 2007.

[11] B. Smith *Phase transition and the mushy region in constraint satisfaction problems.* In Proceedings ECAI'94, Amsterdam, The Netherlands, 1994, pp. 100–104.

[12] Yokoo, M., Durfee, E. H., Ishida, T., Kuwabara, K. *The distributed constraint satisfaction problem: formalization and algorithms.* IEEE Transactions on Knowledge and Data Engineering 10(5), 1998, pp. 673–685.

[13] Yokoo, M., Hirayama, K.*Algorithms for Distributed Constraint Satisfaction: A Review.* Autonomous Agents and Multi-Agent System, 3(2), 2000, pp. 198–212.

[14] U. Wilensky, *NetLogo itself: NetLogo.* Available: http://ccl.northwestern.edu/ netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, 1999.

[15] R. Zivan and A. Meisels, *Dynamic ordering for asynchronous backtracking on Discsps*, Constraints, 11(2-3), 2006, pp. 179–197.

[16] *MAS NetLogo Models-a.* Available: http://jmvidal.cse.sc.edu/netlogomas/.

[17] *MAS NetLogo Models-c.* Available: http://discsp-netlogo.fih.upt.ro/.

Edited by: Dana Petcu and Daniela Zaharie
Received: March 1, 2012
Accepted: April 15, 2012

Scalable Computing: Practice and Experience
Volume 13, Number 1, pp. 73–83. http://www.scpe.org

ISSN 1895-1767
© 2012 SCPE

MULTI-AGENT ARCHITECTURE IN SEMANTIC SERVICES ENVIRONMENT

CRISTINA MINDRUTA,[*] VICTOR ION MUNTEANU,[†] VIOREL NEGRU,[‡] AND CALIN SANDRU[§]

Abstract. A semantic enabled multi-agent architecture for solving non-linear equations systems by using a service oriented approach is proposed. The service oriented approach allows us to access already implemented methods for solving complex mathematical problems. The semantic descriptions of these services provide support for intelligent agents. The proposed architecture is a framework with two extension areas, agent society and domain ontology, with possible application in other domains too.

Key words: multi-agent system, non-linear equations systems, semantic services

1. Introduction. The main goal of this paper is to propose a multi-agent architecture for solving non-linear equations systems. This architecture is designed around a semantic-based solving paradigm supported by an ontology of service descriptions. That allows us to define a multi-agent expert system in a semantic services environment.

Similar work has been done in the MONET project [3] which had the aim to provide a set of web services together with a brokering platform in order to facilitate means of solving a particular mathematical problem. The semantic representation for the mathematical objects was done using OpenMath [20] (MathML [22] was cited also).

GENSS (Grid-Enabled Numerical and Symbolic Services) project [1], like MONET, tries to combine grid computing and mathematical web services using a common open agent-based framework.

In [12] is discussed the matchmaking of semantic mathematical services described using OpenMath.

The architecture we propose is being built based on past experience in designing NESS, a non-linear equations systems solver, and EpODE, an expert system dedicated to ordinary differential equations. NESS [15] is an intelligent front-end for solving non-linear equation systems, developed in CLIPS. Starting from the features of the system to be solved and of the numerical methods, human expert uses domain knowledge (numerical analysis) and heuristics to choose the most suitable method, to interpret the results (intermediary and final), and to restart the solving process in the case of failure. NESS uses task oriented reasoning. A MAS architecture based on UPML has been proposed and instantiated for NESS [18]. EpODE was initially realized as a monolithic expert system [17] and has been re-engineered as a semantic services oriented framework [14]; the solving methodology is workflow-oriented, being realized by integrating semantic services with process modeling.

While having similar main functional objective with NESS, the architecture proposed in this paper is more flexible due to the semantic services component and to the new society of agents designed accordingly.

We have designed a multi-agent architecture that will implement a task-oriented solving model, with a core semantic-based solving paradigm. Typically, multi-agent architecture offers flexibility, scalability and mobility, important quality attributes when dealing with a large number of software services. The agents in the architecture have capabilities ranging from semantically searching for services to providing an execution plan for the given problem. The problem that is to be solved is passed to the multi-agent system as input data. The expert agents in the system analyse the problem and propose an execution plan in order to find the solution. The execution plan is ran and constantly monitored (adjustments are made if needed), and the result of the execution is returned to the user.

The execution plan contains numerical methods which are offered by software services.

We have also designed a semantic services ontology in order to support the semantic-based solving paradigm. For semantic descriptions we have decided to use WSMO (Web Services Modelling Ontology) [2]. In their work, Sorathia et al. [19] analyse several ontologies which were used as an approach to service annotation for discovery, selection, composition, execution and monitoring. Although addressing mainly the challenge of

[*]Department of Computer Science, Faculty of Mathematics and Informatics, West University of Timişoara, Timişoara, Romania (cmindruta@info.uvt.ro).
[†]Department of Computer Science, Faculty of Mathematics and Informatics, West University of Timişoara, Timişoara, Romania (vmunteanu@info.uvt.ro).
[‡]Department of Computer Science, Faculty of Mathematics and Informatics, West University of Timişoara, Timişoara, Romania (vnegru@info.uvt.ro).
[§]Department of Computer Science, Faculty of Mathematics and Informatics, West University of Timişoara, Timişoara, Romania (csandru@info.uvt.ro).

semantic interoperability between relevant service ontologies, we may identify, in this comprehensive literature review providing insight in the state-of-the-art in services ontologies, that WSMO and WSAF [13] are relevant computational ontologies. WSAF is focused on agent mediation and supports dynamic service selection based on QoS (non-functional properties). On the other hand, WSMO is focused on service mediation based both on functional and non-functional properties, allowing more flexibility.

Our approach considers a semantic services context, which is able to offer semantic information useful to the system of agents. In this context, the proposed multi-agent system uses a specific ontology containing concepts, relations and axioms defined for the non-linear equations systems domain, and has an extensible database of semantic descriptions for services implementing numerical methods.

The system implements a core paradigm for solving problems, based on semantic matching between problem properties and numerical method capabilities. Numerical methods are identified based on their semantic descriptions that reflect the properties of the problem for which the method is appropriate. The method selection can be realized by the user based on his own expertise, by the user based on system recommendation and estimations, or automatically by the multi-agent system.

This core paradigm is included in a more flexible approach to solve the problems, that implies coordinated activity in the society of agents and with the user. This can result for example in starting to solve a problem with a numerical method and, from a given step, to continue with another numerical method, based on intermediary results and performance of the system.

Such a flexibility is provided by the proposed multi-agent architecture and covers a large area of user skills, from users with simple mathematical skills, for which the system could provide a solution based on its own expertise, to users with very high mathematical skills, which want to experiment solving new types of problems and using new numerical methods. The experience gained by the later category of users is also captured by the multi-agent system in new methods and new characteristics of the existing ones, thus improving its capabilities.

The paper is structured as follows. Motivation and example use cases are covered in section 2. The conceptual model is covered in section 3. Section 4 is dedicated to the semantic descriptions of the ontology, services and goals used to support the core solving paradigm. Section 5 presents the proposed multi-agent architecture, based on the task-oriented model and integrated with the semantic infrastructure. Conclusions and future work are discussed in section 6.

2. Motivation and use cases. Service Oriented Computing is an emerging computing paradigm in the context of distributed computing. It is already largely accepted and implemented. For example, Cloud Computing is one of the most relevant chapter in service oriented computing. In this context a lot of services in different application domains are available but they are still under-exploited. One of the important challenges is the automation of discovery and composition of services, supported by semantic technologies. The work presented here focuses on the modelling and exploiting the domain of non-linear equation systems, but the presented architecture has flexibility points allowing for its extension to other domains.

We outline here a multi-agent architecture in the context of semantic services intended to provide an expert system to mathematicians. Assuming that different methods for solving non-linear equation systems are implemented in services, the proposed system assists the user in using them and exploring new solving methods.

The following examples of possible use cases for non-linear equations systems solver are gradually more comprehensive.

2.1. Use Case 1 - Solve automatically. The user will provide the non-linear system problem and the system will present the solution. The user can also provide a time limit for finding the solution or the system will use a default value. If the system is not able to find a solution in the imposed amount of time, the user will be informed. The user can establish a new time limit.

2.2. Use Case 2 - Solve under user control. The user will provide the non-linear system problem and the set of session restrictions (time limit, error level, a.s.o.). He can select a numerical method for solving the equations system or can let our system to select one based on its expertise. The user can control the intermediary results and may interrupt the execution to select another numerical method to continue the solving process and/or to modify session restrictions.

2.3. Use Case 3 - Support research. The researcher can add new numerical methods for solving non-linear equations systems. These numerical methods must have been implemented as software services and the researcher adds to the system their semantic descriptions. The new added numerical methods can be then explored inside the previous use cases.

3. Conceptual model.

3.1. Core solving paradigm.
The system implements a core paradigm for solving problems, based on semantic matching between problem properties and numerical method capabilities.

The *domain ontology* describes the following domain:
- *Problem*: A problem is defined by its input data and may get an unique identifier for future recognition in the system. A problem is characterized by a set of properties derived from its input data.
- *Method*: Problems can be solved with methods (numerical methods). A problem can be solved with a numerical method or with a composition of numerical methods. The composition of numerical methods can be predefined or can be created in collaboration with the user. A method is uniquely identified. Besides the problem input data, a method has a specific set of input data. A method is characterized by a set of properties.
- *Session*: A problem is solved under a set of restrictions that define the solving session.
- *Solution*: A solution to a problem is obtained invoking software services.
- *Matching rules*: The expert system recommends numerical methods based on a set of matching rules. The matching rules take into account problem properties and method capabilities.

The *solving paradigm* is focused on matching semantic descriptions, and has the following phases:
- Compute problem properties.
- Identify numerical methods by matching the problem properties with methods capabilities.
- Select numerical method.
- Apply method.

3.2. Task oriented reasoning.
Although sometimes associated with an activity to execute, the concept of task is mostly intended to abstract a specific goal to be achieved [8, 2, 5]. In this regard, tasks definitions do not explicit the particular method to use in order to achieve the goal, but rather give a description of the state of the world to be achieved.

The operational aspect can be abstracted in the concept of a problem solving method (PSM). In their analysis on research on using PSMs in developing Semantic Web applications and based on their practical experience with developing a software using PSMs as conceptual building blocks, the authors in [16] conclude that the fundamental PSM construction techniques used when developing new applications are sufficiently powerful and flexible to build very complex systems.

A PSM describes how to achieve a result based on a set of input data. The goal of a PSM can be regarded as a procedural one in order to obtain a result according to the method specification. The meta-properties of the methods to be mentioned in this context include input, output, precondition, postcondition, sub-task.

The task oriented model is the abstract methodological basis for the proposed non-linear equations system solver, task oriented reasoning being a natural approach for our multi-agent system. Starting from a particular task, one can build a hierarchy of tasks and PSMs that can be considered as the plan for solving the root task. Figure 3.1 represents how may the task-oriented model be applied in the non-linear equations systems solver.

4. Semantic model.
For semantic representations we have adopted Web Services Modelling Ontology. It is a natural approach taking into consideration the fact that, according to [6], WSMO provides a formal ontology for describing Web services based upon WSMF and, in addition, embodies a number of principles that are derived either from the UPML framework, and therefore PSMs research in general, or from the principles underlying the Web and service-oriented computing.

WSMO is a conceptual model that provides ontological specifications for the core elements of semantic Web services. WSMO defines four basic types of elements: ontologies, goals, services and mediators. Ontologies are used to describe application domain ontologies, i.e. entities, relations and constraints in the problem domain. Services are described by non-functional properties, by functionality defined with capabilities, and by behavior defined with interfaces that describe two perspectives: communication and collaboration. Goals have descriptions similar to services, but describe the service requester point of view. Mediators describe mediation entities used to overcome structural, semantic, conceptual disparities.

WSMO is appropriate for modelling the proposed solving paradigm, because it offers a clear separation between goals and services. Services offer numerical methods or compositions of methods from our paradigm,

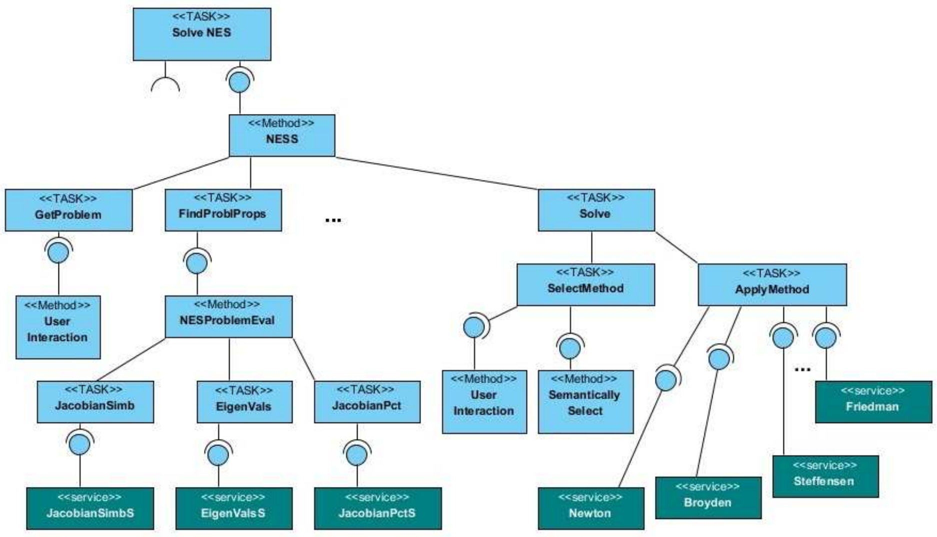


Fig. 3.1: Task-oriented model applied to NESS

and goals are dynamically built for each problem to be solved.

We have used Web Service Modeling Toolkit [9] as support for the modelling activities.

4.1. Knowledge representation. Ontologies. In [14], starting from an existing expert application for solving ordinary differential equation systems, an architecture of services and workflows in a semantic environment have been designed. This was supported by a base ontology for non-linear problems `NonLPOnto`, and a specialization for ODE (ordinary differential equations) systems.

We benefit now from the extensibility of this base ontology to model the non-linear equations systems expert semantics as another extension of this ontology.

`NonLPOnto` represents a pattern for ontologies specific to different non-linear problem categories. One of these categories is represented by the non-linear equations systems (NES). As with ODE systems, the main concepts defined in `NESOnto` are specializations of concepts defined in `NonLPOnto`(figure 4.1).

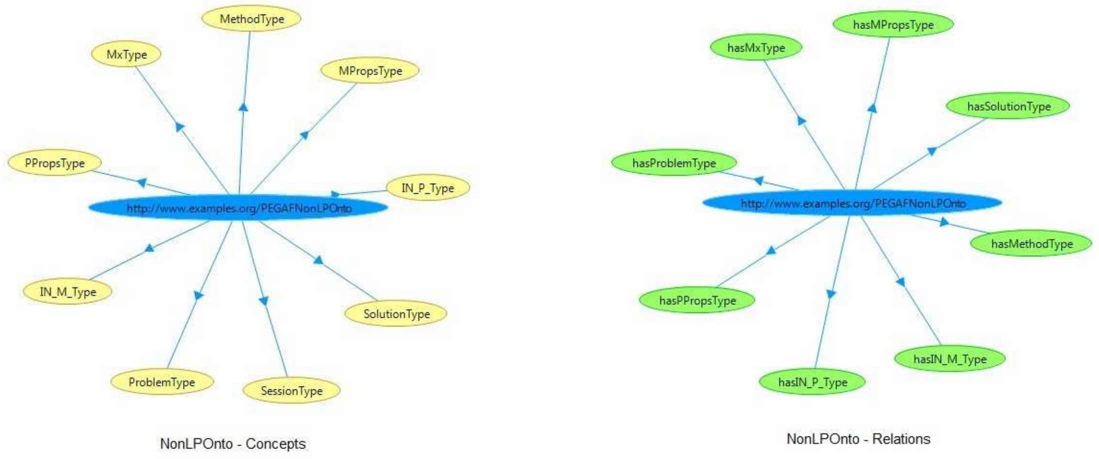


Fig. 4.1: `NonLPOnto` represented with WSMT

`NonLPOnto` is designed as a high level WSMO ontology for modeling mathematical services for solving non-linear problems. According to this model a problem has an input data set and a set of properties. An associated matrix is build based on the input data set, and the characteristics of this matrix are used to specify properties of the problem. A problem is solved in the context of a session that defines specific requirements for computation and result. Solving a problem implies applying a numerical method. A numerical method has also a specific input set and is characterized by a set of properties.

Refining things, `NESOnto` (figure 4.2) contains concepts, relations and axioms that define problem and solution spaces of the non-linear equations systems.

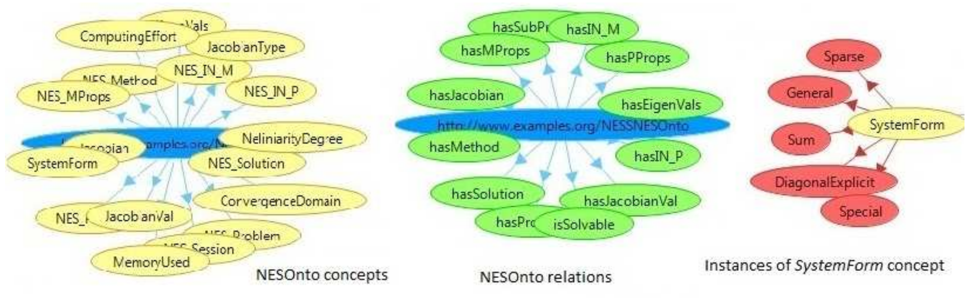


Fig. 4.2: Elements of `NESOnto` represented with WSMT

`NESOnto` defines the concept of problem (`NES_Problem`) in relation to the concepts representing the input data of the problem (`NES_IN_P`) and the properties of the corresponding non-linear equations system (`NES_PProps`).

It also defines the concept of numerical method (`NES_Method`) in relation to the concepts representing the input data of the numerical method (`NES_IN_M`) and the properties of the numerical method (`NES_MProps`). In figure 4.3, the concept `NES_MProps` is represented with its attributes and the relation `hasPProps` between the `NES_Method` and its properties.

The restrictions imposed on the solving session are modelled with the concept `NES_Session` and the solution of the non-linear equations system is modelled with the concept `NES_Solution`.

The associated matrix is modelled with two concepts: `Jacobian` represents the symbolic Jacobian of the system, and `JacobianVal` represents the Jacobian matrix of the system computed in a given point.

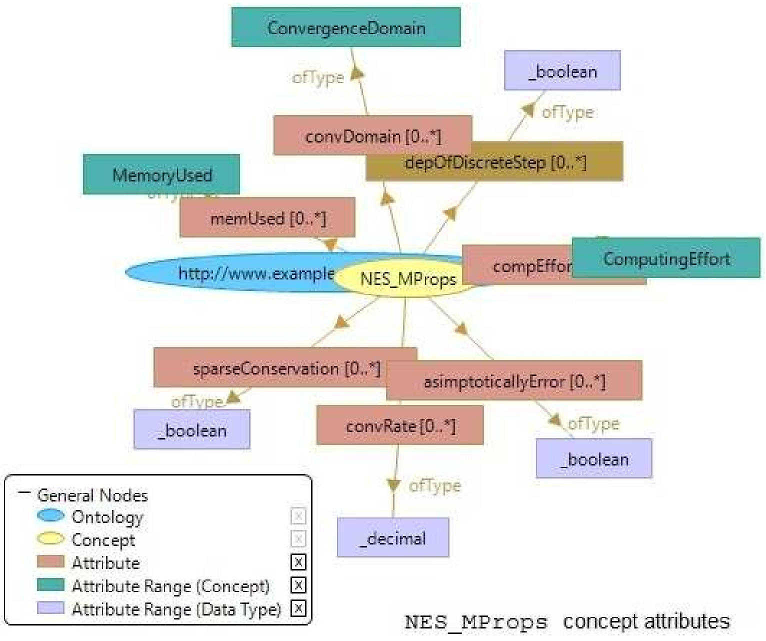
The properties of the non-linear equations system are of types defined in specific concepts (ex. `SystemForm`), and for each of these concepts the particular instances (ex. `General`, `Sparse`, `DiagonalExplicit`) have been defined.

The properties of the numerical method have been defined in the same manner. They will be used by the expert agent, that will refine the service selection and composition matching them to the execution constraints represented as an instance of the `NES_Session` concept.

One key element of `NESOnto` is the relation *isSolvable* applied to the problem properties. The relation is used in the matching process between the problem characterized by its set of properties and a numerical method which could solve the specific problem. An instance of this relation is created when a semantically described service can solve the problem.

4.2. Services implementing numerical methods. The capabilities of each service are described using `NESOnto` and an ontology specific to the service that contains one or more axioms. One of the axioms in this specific ontology defines the relation *isSolvable* by expressing the properties of the non-linear equations systems for which the numerical method is appropriate.

In figure 4.4(a) is represented the semantic description of `Broyden_NES` service that implements the numerical method Broyden for solving non-linear equations systems. The precondition in the semantic description states that the method can be used if the relation *isSolvable* exists for the properties of the problem to be solved. The semantics of this relation for the service `Broyden_NES` are defined in the axiom *isSolvableDef* of the ontology particular to the capabilities of this service, and expresses the fact that Broyden method is recommended for non-linear equations systems of general form, with non-singular Jacobian, and of medium (between 10 and 50



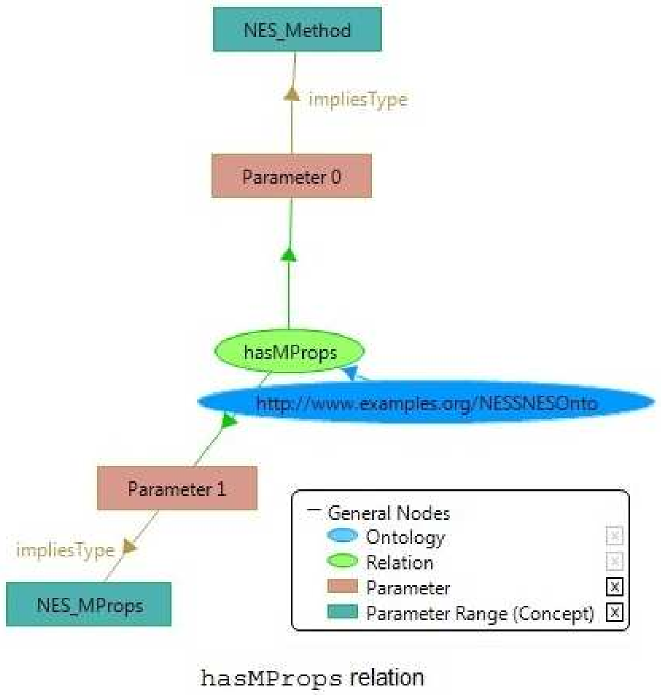
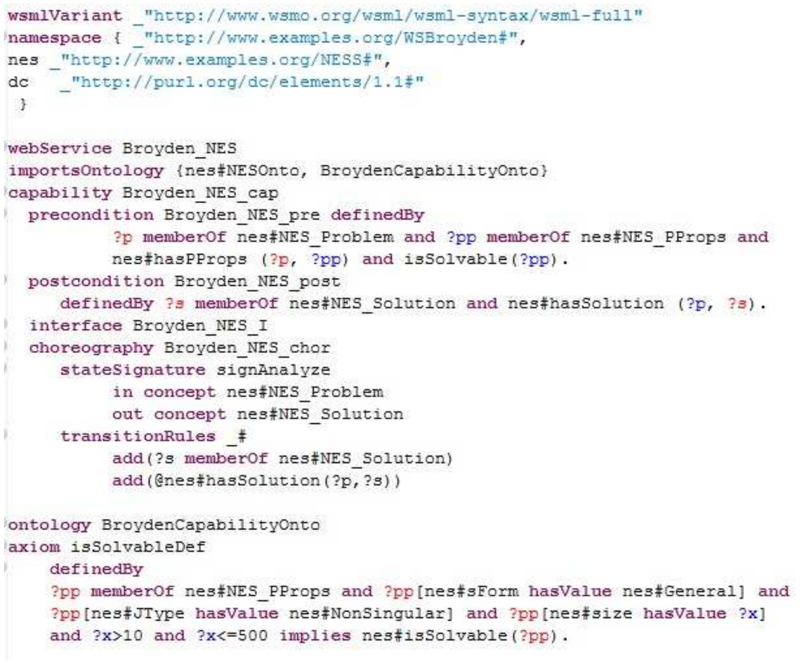


Fig. 4.3: Method and its properties represented with WSMT

equations) or big size (between 50 and 500 equations). This represents a part of the expert knowledge and is implemented in the semantic description of the service.

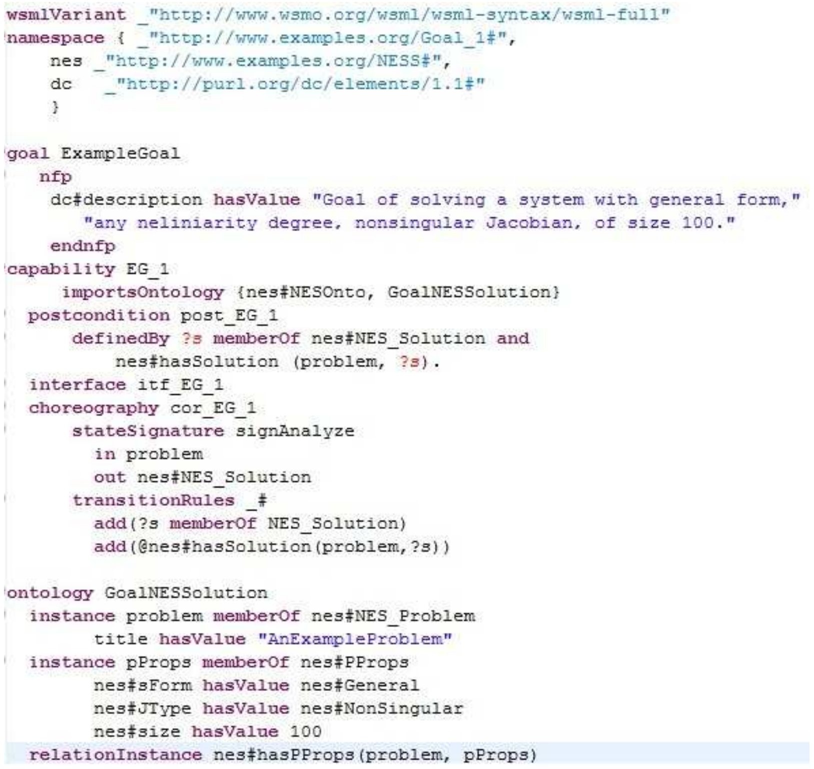


```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-full"
namespace { _"http://www.examples.org/WSBroyden#",
nes _"http://www.examples.org/NESS#",
dc   _"http://purl.org/dc/elements/1.1#"
 }

webService Broyden_NES
importsOntology {nes#NESOnto, BroydenCapabilityOnto}
capability Broyden_NES_cap
  precondition Broyden_NES_pre definedBy
          ?p memberOf nes#NES_Problem and ?pp memberOf nes#NES_PProps and
          nes#hasPProps (?p, ?pp) and isSolvable(?pp).
  postcondition Broyden_NES_post
     definedBy ?s memberOf nes#NES_Solution and nes#hasSolution (?p, ?s).
  interface Broyden_NES_I
  choreography Broyden_NES_chor
     stateSignature signAnalyze
          in concept nes#NES_Problem
          out concept nes#NES_Solution
     transitionRules _#
          add(?s memberOf nes#NES_Solution)
          add(@nes#hasSolution(?p,?s))

ontology BroydenCapabilityOnto
axiom isSolvableDef
    definedBy
    ?pp memberOf nes#NES_PProps and ?pp[nes#sForm hasValue nes#General] and
    ?pp[nes#JType hasValue nes#NonSingular] and ?pp[nes#size hasValue ?x]
    and ?x>10 and ?x<=500 implies nes#isSolvable(?pp).
```

(a)



```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-full"
namespace { _"http://www.examples.org/Goal_1#",
    nes _"http://www.examples.org/NESS#",
    dc   _"http://purl.org/dc/elements/1.1#"
    }

goal ExampleGoal
   nfp
    dc#description hasValue "Goal of solving a system with general form,"
       "any neliniarity degree, nonsingular Jacobian, of size 100."
    endnfp
capability EG_1
     importsOntology {nes#NESOnto, GoalNESSolution}
  postcondition post_EG_1
      definedBy ?s memberOf nes#NES_Solution and
          nes#hasSolution (problem, ?s).
  interface itf_EG_1
  choreography cor_EG_1
      stateSignature signAnalyze
        in problem
        out nes#NES_Solution
      transitionRules _#
        add(?s memberOf NES_Solution)
        add(@nes#hasSolution(problem,?s))

ontology GoalNESSolution
  instance problem memberOf nes#NES_Problem
        title hasValue "AnExampleProblem"
  instance pProps memberOf nes#PProps
        nes#sForm hasValue nes#General
        nes#JType hasValue nes#NonSingular
        nes#size hasValue 100
  relationInstance nes#hasPProps(problem, pProps)
```

(b)

Fig. 4.4: (a) WSML descriptions for `BroydenNES` service and (b) `ExampleGoal` goal

4.3. Goals. Goals are dynamically built for each problem. Each goal has its a particular ontology that contains instances of concepts from `NESOnto`. In order to identify the services which are able to solve the corresponding problem, the particular ontology (`GoalNESSolution`) contains an instance of the `NES_PProps` concept which holds the concrete properties of the given problem.

In figure 4.4(b), a goal of solving a non-linear equations systems is described in WSML.

5. Multi-agent architecture. Using a multi-agent system for service discovery and composition enables a decentralized approach to solving non-linear equation systems. This approach is further enhanced by an event driven model and concurrency offered by the agents.

When developing the multi-agent architecture, we had several architectural concerns in mind. The architecture must:

- Allow service operations: publishing, semantic facilitation, invoking (running) etc.
- Provide automatic semantic service selection and composition.
- Detect and recover from a failing service.
- Monitor services.
- Create solving scenarios based on previous solving experience.

In our architecture, the PSMs can be implemented as semantic services or as agents (figure 5.1).

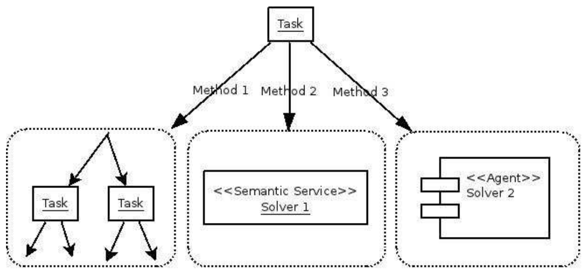


Fig. 5.1: Task structure

5.1. Agents. The multi-agent system is composed of the following agents: service, monitoring, execution, user interface, matchmaking, reasoning and historian. The proposed architecture is depicted in figure 5.2.

The user interface agent handles all communication with the users and provides users with proxy functionality for interacting with the system. User interface agent exposes a REST interface to which users can connect. It communicates with the reasoner, executioner and monitoring agents in order to manage the planning and execution.

The reasoner agent is in charge of creating task-oriented plans to solve the problems it receives from the human interface agent. It uses the domain ontology to create the semantic definitions of the tasks in terms of WSMO goals by processing the input data in order to detect problem properties. Furthermore, the historian agent is used to identify plans that where applied in similar problems. After the plan has been made, it is dispatched to one of the execution agents for processing. Similarly to the WSMX platform, the reasoner agent will use one of the following reasoners: IRIS[1], KAON2[2], PELLET[3] or MINS[4].

The execution agent handles the execution of the work plan. It will be in charge of service invocation and workflow management. The execution agent will query the matchmaking agent to retrieve information like endpoints for semantically compatible services for the current work plan. Service invocation is done through service agents. Also, synchronization with monitoring agents will be done in order to monitor current work plan progress. In case of execution problems, it will ask a reasoner agent to find alternatives/suggest a solution for it.

The monitoring agent has the role of monitoring the current task execution. It will synchronize with execution agents for work plan information. It will also send periodic updates to the user interface agents

[1]http://www.iris-reasoner.org/
[2]http://kaon2.semanticweb.org/
[3]http://clarkparsia.com/pellet
[4]http://tools.sti-innsbruck.at/mins/

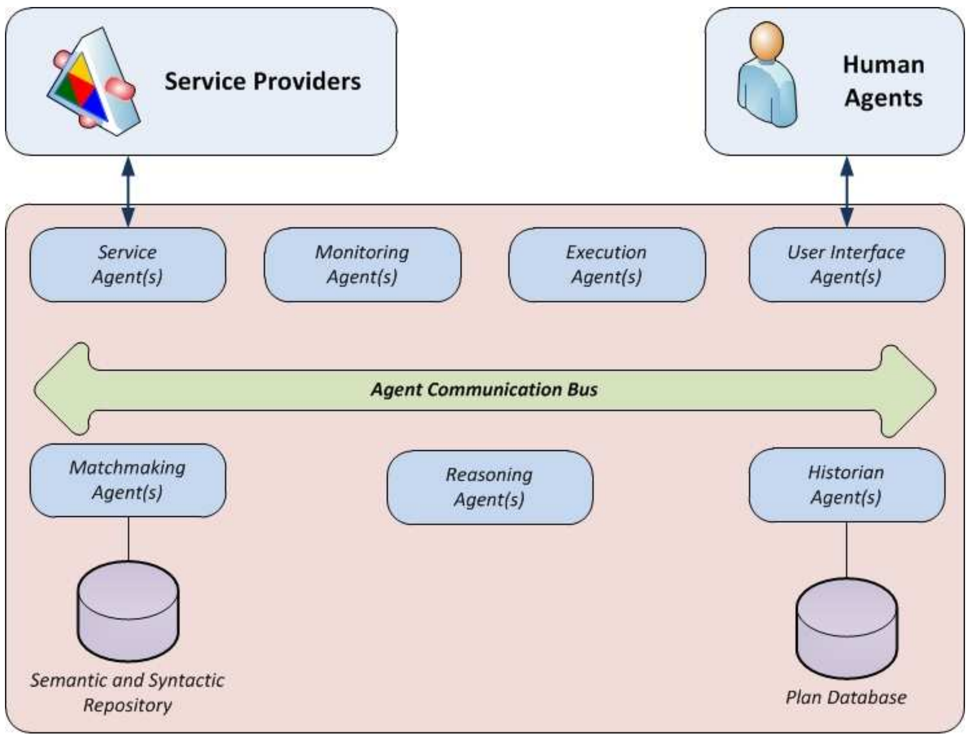


Fig. 5.2: Multi-agent system architecture

so they can relay task progress to the user. Monitoring agents also have the capability of creating execution profiles that they will end to historian agents. These profiles cover the running work plan along with the service decisions made and any errors that may have appeared.

The historian agent is in charge of archiving data and making it available for reasoner agents. It receives problem profiles from monitoring agents and stores them in a database. Storing these profiles is important because they contain information about previous task executions, information that is important for reasoner agents to make decisions.

Service agents have the role of communication with service providers. They can invoke services and also play a proactive role in the system by discovering services that can be integrated within the system. They will look in WSIL[5]/UDDI[6] service repositories in order to identify compatible services based on the ontology.

Last but not least, the matchmaking agents have two responsibilities. One is to handle storing semantic descriptions of domain ontology and of services, and WSDL descriptions of services. This will make the service database available for the system while alleviating the burden of dealing with storage mechanisms. The other responsibility is to offer goal-service matching functionality. Mainly, the execution agent can directly ask for a specific service or can ask for a proper service to be found by performing semantic matching between the service and the goal descriptions. In the proposed architecture we have focused on the relevant components for proving the concepts involved in a dynamic support for solving non-linear equations systems in a semantic services environment. We assumed the existence of the needed content in the Semantic and Syntactic Repository describing web services. Creation and maintenance of this repository is a complex task that will be considered in a future work.

One important component of this multi-agent architecture is the agent communication bus. This bus enables synchronous and asynchronous calls between the agents and is available in a distributed setting.

Figure 5.3 depicts the interaction between agents, interaction that leads to finding a solution for a simple problem.

[5]http://www.ibm.com/developerworks/library/specification/ws-wsilspec/
[6]http://uddi.xml.org/

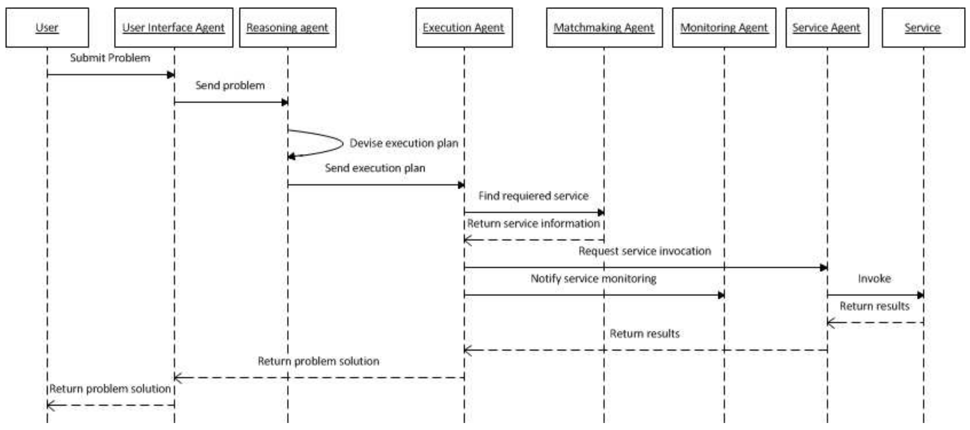


Fig. 5.3: An example of a simple problem solving sequence

5.2. Agent platform. Our approach in implementing the multi-agent system is based on the Akka platform[7]. Akka, while not being a traditional, FIPA compliant [4], multi-agent platform like Jade[8], offers tools to build highly concurrent, distributed, and fault tolerant event-driven applications [21]. Akka allows applications to be written both in Java[9] and in Scala[10] while implementing the actor model. The actor model enables our agents to have reactive behavior in solving the tasks.

Akka's integration with Apache Camel[11] allows agents to send and receive messages using a large number of protocols and APIs. Also, support for the Spring Framework[12] enables faster development.

Matchmaking is an important functionality of such a system. Different approaches are offered by WSMX [7, 11], WSMO-MX [10] or are domain specific and user oriented [23]. We have adopted a simplified version which makes use of a particular pattern in semantic descriptions of the services.

5.3. Framework and extensibility. The proposed architecture has a great flexibility due to the event-driven architectural style of the infrastructure and to the inherent flexibility of multi-agent systems. It can be seen as a framework with two main extensibility areas. The first extensibility area is in the society of agents which can be extended with new types of agents that augment and refine the solving paradigm. The second extensibility area stands in the semantic definitions. Extending the ontology allows to cover more mathematical chapters. Changing the ontology results in adapting to different domains where problems can be solved based on a similar core solving paradigm. Moreover, the two types of extensions can be combined.

6. Conclusions and future work. The task-oriented model makes a clear separation between tasks to be accomplished and methods for solving them. This model is implemented using a multi-agent architecture and is applied to design a solver for non-linear equations systems.

The multi-agent system is integrated with services implementing numerical methods. The services are semantically described in terms of a domain ontology we propose for non-linear equations systems.

Semantic descriptions are realized in WSML, allowing us to benefit from the clear separation between goals and services. This maps over the task-oriented model, specifically over tasks and solving methods respectively.

Specialized agents dynamically build semantic descriptions of the goals based on the properties of problems to be solved, and appropriate services are discovered by semantic matching.

Other agents are implied in user interaction, in building work plans, in managing the system, aiming to offer solutions to different problems or support for the human expert to experiment numerical methods in solving non-linear equations systems.

The multi-agent system is build over Akka platform. The system will be validated in the context of NESS.

[7]http://akka.io/
[8]http://jade.tilab.com/
[9]http://www.oracle.com/us/technologies/java/overview/index.html
[10]http://www.scala-lang.org/
[11]http://camel.apache.org/
[12]http://www.springsource.org/

Our future work has more directions: to extend the definitions of the knowledge in the domain of non-linear equations systems and to combine WSML with OpenMath and MathML; to include the chapter of ordinary differential equations systems and possible other mathematical chapters; to open the framework towards other domains.

Acknowledgments. This work was partially supported by the Romanian Government PNII grant nr. 12118/2008 (SCIPA), by POSDRU/88/1.5/S/49516 structural funds grant, ID 49516 (2009), and by the grant POSDRU 21/1.5/G/13798.

REFERENCES

[1] *Grid-enabled numerical and symbolic services. [Online] Available: http://genss.cs.bath.ac.uk/.*

[2] *Web service modeling ontology. [Online] Available: `http://www.wsmo.org/`.*

[3] M. Aird, W. Medina, and J. Padget, *Monet: service discovery and composition for mathematical problems*, in Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on, may 2003, pp. 678 – 685.

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

[5] B. Chandrasekaran, *Design problem solving: a task analysis*, AI Mag., 11 (1990), pp. 59–71.

[6] J. Domingue and D. Fensel, *Problem solving methods in a global networked age*, Artif. Intell. Eng. Des. Anal. Manuf., 23 (2009), pp. 373–390.

[7] D. Fensel, F. M. Facca, E. Simperl, I. Toma, D. Fensel, F. M. Facca, E. Simperl, and I. Toma, *The web service execution environment*, in Semantic Web Services, Springer Berlin Heidelberg, 2011, pp. 163–216.

[8] D. Fensel, E. Motta, F. V. Harmelen, V. R. Benjamins, M. Crubezy, S. Decker, M. Gaspari, R. Groenboom, W. Grosso, M. Musen, Enric, E. Plaza, G. Schreiber, R. Studer, and B. Wielinga, *The unified problem-solving method development language UPML*, Knowledge and Information Systems, 5 (1999), p. 2003.

[9] M. Kerrigan, A. Mocan, E. Simperl, and D. Fensel, *Modeling semantic web services with the web service modeling toolkit*, Journal of Network and Systems Management, 17 (2009), pp. 326–342. 10.1007/s10922-009-9130-8.

[10] M. Klusch and F. Kaufer, *Wsmo-mx: A hybrid semantic web service matchmaker*, Web Intelli. and Agent Sys., 7 (2009), pp. 23–42.

[11] S. Komazec and F. Facca, *Whats new in wsmx?*, in The Semantic Web: Research and Applications, L. Aroyo, G. Antoniou, E. Hyvnen, A. ten Teije, H. Stuckenschmidt, L. Cabral, and T. Tudorache, eds., vol. 6089 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2010, pp. 396–400.

[12] S. Ludwig, O. Rana, J. Padget, and W. Naylor, *Matchmaking framework for mathematical web services*, Journal of Grid Computing, 4 (2006), pp. 33–48. 10.1007/s10723-005-9019-z.

[13] E. Maximilien and M. Singh, *A framework and ontology for dynamic web services selection*, Internet Computing, IEEE, 8 (2004), pp. 84 – 93.

[14] C. Mindruta and D. Petcu, *A semantic services architecture for solving ODE systems*, in Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2010 12th International Symposium on, sept. 2010, pp. 301 –307.

[15] V. Negru, S. Maruster, and C. Sandru, *Intelligent system for non-linear simultaneous equation solving*, in Technical Report Report Series. No, 98-19. RISC-Linz, december 2003.

[16] M. j. O'connor, C. Nyulas, S. Tu, D. l. Buckeridge, A. Okhmatovskaia, and M. a. Musen, *Software-engineering challenges of building and deploying reusable problem solvers*, Artif. Intell. Eng. Des. Anal. Manuf., 23 (2009), pp. 339–356.

[17] D. Petcu, *Expert system for ordinary differential equations. [Online] Available: http://www.info.uvt.ro/ petcu/epode/-main.htm.*

[18] C. Sandru and V. Negru, *Validating UPML concepts in a multi-agent architecture*, in Schedae Informaticae, vol. 15, 2006, pp. 109–126.

[19] V. Sorathia, L. Ferreira Pires, and M. S. van, *An analysis of service ontologies*, Pacific Asia Journal of the Association for Information Systems, 2 (2010), pp. 17–46.

[20] The OpenMath Society, *Openmath. [online] available: http://www.openmath.org/.*

[21] Typesafe Inc, *Akka Documentation Release 2.0. [Online] Available: `http://doc.akka.io/docs/akka/2.0/Akka.pdf`*, March 2012.

[22] W3C, *Mathml. [online] available: http://www.w3.org/math/.*

[23] M. Wilkinson, B. Vandervalk, and L. McCarthy, *The semantic automated discovery and integration (sadi) web service design-pattern, api and reference implementation*, Journal of Biomedical Semantics, 2 (2011), p. 8.

Edited by: Dana Petcu and Daniela Zaharie
Received: March 1, 2012
Accepted: April 15, 2012

Scalable Computing: Practice and Experience
Volume 13, Number 1, pp. 85–98. http://www.scpe.org
ISSN 1895-1767
© 2012 SCPE

A RING-BASED PARALLEL OIL RESERVOIR SIMULATOR*

LEILA ISMAIL†

Abstract. We develop and implement a ring-based parallel 3-D oil-phase homogeneous isotropic reservoir simulator and study its performance in terms of speedup as a function of problem size. The ring-based approach is shown to result in significant improvement in speedup as the problem size increases. This improvement stems from the reduction in communication costs inherent in a ring-based approach. The simulator employs a parallel conjugate gradient (CG) algorithm that we develop for solving the associated system of linear equations. The parallelization uses an MPI programming model. Previously proposed parallel oil reservoir simulators focus on data parallelism and load balancing and gives less attention to the communication cost. Performance analysis is given showing that the parallel algorithm results in a speedup of more than 42 times compared to a sequential simulator for a large simulation problem. This major improvement occurs for larger problem sizes, since the communication savings become significant. We compare our results to the implementation of the parallel oil reservoir simulator using the Portable Extensible Toolkit for Scientific Computation (PETSc). Oil reservoir simulators are used for forecasting reservoir potential before costly drilling, and are essential for improving oil recovery from existing fields, helping to maximize oil production. The speedup gained through the technique presented here can result in major savings of engineering time and more accurate reservoir management, and in turn higher oil production. Existing simulators suffer from limited performance due to the huge numerical operations involved. To cope with the issue, engineers usually reduce the size of the simulation model to get results in an acceptable timeframe, sacrificing accuracy of the predictions. This article describes the proposed ring-based algorithm for parallelization and development of a 3-D oil phase reservoir simulator. The work is a prelude to further planned research to develop an extended simulator that applies to three phases (oil, gas, and water) and to a heterogeneous and non-isotropic.

Key words: Parallel Computing, Oil Reservoir Simulator, Conjugate Gradient Method, Performance Evaluation

1. Introduction.

Oil reservoir simulators ([1]- [2]) are important tools in the petroleum industry. They help decision makers in oil reservoir forecasting, analysis, history matching, and recovery. To correctly make decisions regarding recovery of hydrocarbons, an accurate numerical model of the reservoir must be established to predict outcomes and performance under various operating conditions. These include location and rate of injection in wells, and the recovery techniques, the selection of which has a great impact on oil field operation from a financial perspective. It is well known that the accuracy of a simulation depends upon the *resolution* used. Finer resolutions of discretization grids improve the accuracy of the simulation [3], but are accompanied by an increase in problem size. Therefore, the CPU time required for the simulator to run increases considerably with granularity due to the huge number of equations that must be included in the model. The conjugate gradient method (CG) (initiated in [4]; see also [5]) is one of the best known and most powerful iterative linear system solvers used in many simulation problems.

There is a great interest in parallelizing oil reservoir simulators to increase simulators' precision and consequently oil production. On one hand, existing parallel oil reservoir simulators focus on data parallelism and load balancing and gives less attention to the resulting communication cost. They often use domain decomposition techniques and parallel solvers which in turn use matrix decomposition techniques and give less attention to the resulting communication overhead. On the other hand, the parallelization may involve the numerical representation of the field, resulting in a change in the numerical equations.

In this work, we develop a parallel oil reservoir simulator which uses a parallel CG method to solve its associated system of linear equations. Our parallel oil reservoir simulator has the following advantages:

- Preserving the oil reservoir simulator numerical equations. Our parallel simulator focus on reducing communication overhead generated from data parallelization of the reservoir without any change in the numerical representation of the oil reservoir simulator. This allows the portability of our parallel solution to many of the existing oil reservoir simulators, as opposed to a parallelization approach which introduces a change in the numerical representation ([6], [7], [8], [9]).
- Scalability. Our parallel implementation scales well with increasing problem size and increasing number of computing resources. For instance, [10] used a parallel CG (using row-wise distribution of the coefficient simulation matrix) for a 3-D, 3-phase oil reservoir simulator and obtained a speedup of nearly 4.5 using 32 processors for a medium size problem (19,584 equations) using CS-2 with 100 MHz HyperSPARC architecture (see Figure 3 in [10]). Our parallel implementation in our experimental environment leads to a similar speedup for a Class A problem size (14,000 equations) and to a speedup

*This work was supported by the United Arab Emirates University

†Faculty of Information Technology, United Arab Emirates University, P.O.Box 17551, Al-Ain, UAE. (leila@uaeu.ac.ae).

of 23.63 for Class C problem (150,000 equations), indicating that our parallel approach becomes more effective as the problem size increases.

- Reducing communication overhead. Our scheme uses the ring-based technique to reduce communication overhead generated from parallelization. Parallel oil reservoir simulator needs to exchange data between its parallel computational elements.
- Load balancing. To maximize the parallel computing efficiency of the simulator and the use of the parallel computing environment, i.e., the processors involved in the computation, it is important to have a load balanced distribution of the computational load. We implement a load-balanced distribution of the CG computation among the processors involved in the parallel computation. This is obtained by using the greedy approach in distributing the reservoir coefficients among the available processors.

For an efficient parallel oil reservoir simulator, it is essential to have a scalable approach with increasing problem size and increasing number of processors. In this work, we propose a parallel algorithm that is optimized by a ring-based approach to reduce communication cost. The ring-based approach is a known technique to reduce communication cost [11]. Combing this technique with the data decomposition approach has led to a speedup of 42 times in our experimental environment using 128 computing processors for large problem size (Class C). However, to our knowledge, the technique has not been brought to bear on oil applications. This result is promising in the oil industry as it can save significant engineering time and facilitate more accurate reservoir management. We apply the ring-based approach to achieve parallelism and communication in multiple steps. We use a distributed memory environment consisting of 128 cores of Intel Xeon 5355 to evaluate the performance of the parallel 3-D oil-phase reservoir simulator. To maximize the parallel computing efficiency of the simulator and the use of the parallel computing environment, i.e., the processors involved in the computation, it is important to have a load balanced distribution of the computational load. We implement a load-balanced distribution of the CG computation among the involved processors. Our parallelization technique is evaluated by measuring the speedup gain of our parallel simulator compared to the scalar sequential, the former being demanding in terms of design efforts. We compare our results to a parallel implementation using the Portable Extensible Toolkit for Scientific Computation (PETSc) [12]. PETSc is a suite of data structures and routines for the parallel solution of scientific applications modeled by partial differential equations. It is widely used in many scientific simulations including oil reservoir simulator ([13], [14]). PETSc implements row-wise distribution of matrices and does not consider a load-balanced approach. It uses a one-step overlapping mechanism, in which a matrix is divided into submmatrices; processors send data asynchronously and start computing with diagonal submatrices in parallel, hoping that global data is collected meanwhile the local computation is taking place to continue with the remaining submatrices [15].

The rest of the paper is organized as follows. Section 2 overviews related works. In section 3, we describe the oil reservoir model's partial differential equations. In section 4, a numerical model for the reservoir is presented. The programming model including the parallel approach and its implementation are described in section 5. Our experiments and the associated results are presented in section 6. Concluding remarks are given in section 7.

2. Related Works. Several works developed parallel oil reservoir simulators on a distributed memory environment. Many of the existing parallel oil reservoir simulators rely on parallel library routines for parallel distribution of their linear solvers. For instance, in [10], the parallel oil reservoir simulator implements a parallel Conjugate Gradient method which uses the Sparse Distributed Data Library (DDL), for a 3-D, 3-phase oil reservoir simulator. The DDL implements a row-wise distribution of the coefficient simulation matrix. The implementation r for the implementation of a parallel CG linear solver uses all-to-all communications technique. By using this technique, the size and the number of messages exchanged between the different computing processors increase with increasing problem size and number of processors, inducing scalability issues. Figure 3 in [10] shows a speedup of nearly 4.5 using 32 processors for a medium size problem (19,584 equations) using CS-2 with 100 MHz HyperSPARC architecture. Reference [14] relies on PETSc library [12] which implements parallel linear solvers based on the library parallel routines. Reference [16] developed a parallel oil reservoir simulator based on the overlapping domain decomposition, in particular Additive Schwarz with Overlap linear solver, and the parallel Singular Value Decomposition linear solver, using multi-core multi-processor shared-memory (SMP) desktops, and obtained a speedup of 1.6 on 2 CPUs and of 1.7 on 4 CPUs. Reference [17] developed a parallel oil reservoir simulator based on domain decomposition and parallelized its underlying linear solver; i.e., the strongly implicity procedure (SIP). The simulator is implemented using MPI on CRAY T3E system and IBM SP2 systems. The speedup obtained was 14 using 80 processors on CRAY T3E system, and 30.6 using

80 processors on IBM SP2 system. However, on IBM SP2 system, the performance did not scale beyond 62 processors with increasing number of processors.

Other works involves a change in the numerical solution to incorporate more parallelism. For instance, reference [8] is based on the constrained pressure residual (CPR) a multi-stage parallel linear solver [9], and the ILU0 parallel iterative solver. Figure 4 in reference [8] indicates a speedup of 28 using the CPR solver and a speedup of 12 using the ILU0 on 64 processors for a 3-D, an incompressible water oil 2-phase for 1,094,721 grid blocks; a very large problem size. Reference [18] tested combination of multiscale (in time and in space) simulation and compared them to single-spatial dual-temporal simulations and concluded that the best combination is the dual-spatial dual-temporal. References [6] and [7] rewrite the Conjugate Gradient method linear solver algorithm into blocks of algorithms to reduce synchronization between among its iterations and therefore communication cost, and consequently incorporate more parallelism.

Several algorithms have been published for parallelizing CG as a standalone application [19], [20], [21], [22]. They are developed for general-purpose engineering applications and are not tailored to oil reservoir modeling. In [19] and [20], algorithms have been implemented on a specialized event-driven multi-threaded platform. In [21] and [22], algorithms have been implemented on a distributed shared memory cluster. Field [23] optimizes CG for regular sparse matrices and studies the impact of mesh partitioning on the performance. In references S [24] and [25], the authors introduce data decomposition strategies for CG on hypercubes and mesh networks for unstructured sparse matrices. Blocks of matrices are assigned to processors to achieve a partial result of the matrix-vector multiplication in the CG algorithm. In reference [24], a ring-based overlap mechanism is used for global summation within the CG method, a speedup of of 2.5 was obtained on 128 cores compared to the original National Aeronautics and Space (NAS) benchmark [26] on Intel iPSC/860 hypercube architecture. Reference [11] presents communication-avoiding algorithms to decrease communication costs of applications. Reference [20] used also the ring-based algorithm for the CG method as a standalone application and obtained a speedup of 41 on 65 processors of type ChibaCity. We obtained almost the same speedup [20] for big problem size (Class C matrix size), though [20] performed measurements on unstructured matrices with more non-zeros than our heptadiagonal matrices, thus more computations are involved to overlap communication cost.

3. 3-D Oil-Phase Reservoir Partial Differential Equation Model.

Development of a parallel reservoir simulator includes the following steps:

- Develop the partial differential equations of the model based on the oil reservoir characteristics. For the 3-D oil-phase reservoir model, the equations have one unknown variable, namely pressure.
- Divide the oil reservoir into grids and discretize the partial differential equations in space and time. In the case of a homogeneous and isotropic reservoir, the discretization of the equations produces a linear system of equations.
- Determine an ordering scheme from stencils to obtain an order of the coefficients of the linear system of equations and choose a linear solver which will be used to find the solution; i.e., the pressure per grid element of the oil reservoir. In case of a homogeneous isotropic oil reservoir, all the coefficients are constants in space and time.
- Parallelize the model and code it.
- Test the simulator by comparing results it gives to known results obtained from another proven simulator.

The partial differential equations reflect the reservoir characteristics, such as the reservoir boundaries, rock properties including porosity and permeability, and well production and injection data input [1], [27]. In this study, we consider a simulator for a 3-D homogeneous and isotropic oil-phase reservoir. The differential equations of the reservoir model are derived from Equations 3.1, 3.2, and 3.3. Equation 3.1 is Darcy's law. It represents a relationship between the field velocity u and the field pressure p. Equation 3.2 is a statement of mass balance. Equation 3.3 represents the formation volume factor per bulk volume of the reservoir:

$$u = -\beta_c \frac{k}{\mu} \left(\nabla p - \gamma \nabla Z\right) \tag{3.1}$$

$$-\frac{\partial}{\partial x}(\dot{m}_x) - \frac{\partial}{\partial y}(\dot{m}_y) - \frac{\partial}{\partial z}(\dot{m}_z) = \frac{\partial(m_v)}{\partial t} - q_s \tag{3.2}$$

$$B_o = \frac{\rho_o sc}{\rho_o} \tag{3.3}$$

Here β_c is a unit conversion factor for the permeability coefficient, k is the rock permeability, μ is the dynamic viscosity of the fluid, Z is the elevation (positive in downward vertical direction), and γ is the fluid gravity, which is the fluid density in terms of pressure per distance. $(\dot{m})$ denotes the mass flow rate per unit of time and per unit area, q_s is the mass density source or sink (mass per unit of time), m_v represents the mass of fluid contained in a unit of volume of the reservoir. B_o is a formation volume factor which is the ratio of the density of the oil at standard conditions $(\rho_o sc)$ to the density of the oil at reservoir pressure and temperature ρ_o. Standard conditions are usually 60^oF and $14.7psia$ in oil fields [27].

Mass flow rate is expressed as the product of the oil density (ρ) and Darcy's velocity (u). The mass per unit volume (v_m) is represented by the product of oil density and porosity (ϕ). The mass flow rate q_s is formulated as the product of the fluid density and volumetric flow rate q. Then we have the following formulas [1]:

$$\dot{m}_x = \alpha_c \rho u_x \tag{3.4}$$

$$\dot{m}_y = \alpha_c \rho u_y \tag{3.5}$$

$$\dot{m}_z = \alpha_c \rho u_z \tag{3.6}$$

$$m_v = \rho\phi \tag{3.7}$$

$$q_s = \alpha_c \rho q \tag{3.8}$$

Based on Equations 3.4, 3.5, 3.6, 3.7, 3.8 and 3.3, Equation 3.2 becomes

$$-\frac{\partial}{\partial x}\left(\frac{u_x}{B_o}\right) - \frac{\partial}{\partial y}\left(\frac{u_y}{B_o}\right) - \frac{\partial}{\partial z}\left(\frac{u_z}{B_o}\right) = \frac{1}{\alpha_c}\frac{\partial}{\partial t}\left(\frac{\phi}{B_o}\right) - q_{sc} \tag{3.9}$$

The equation 3.9 involves two unknowns: the velocity field and the porosity. The closure model used to complete the model is the Darcy's law (Equation 3.1). For simplicity, we assume negligeable gravital forces. Equation 3.9 becomes:

$$\frac{\partial}{\partial x}\left(\beta_c \frac{k_x}{\mu B_o}\frac{\partial p}{\partial x}\right) + \frac{\partial}{\partial y}\left(\beta_c \frac{k_y}{\mu B_o}\frac{\partial p}{\partial y}\right) + \frac{\partial}{\partial z}\left(\beta_c \frac{k_z}{\mu B_o}\frac{\partial p}{\partial z}\right) + q_{sc} = \frac{1}{\alpha_c}\frac{\partial}{\partial t}\left(\frac{\phi}{B_o}\right) \tag{3.10}$$

where B_o is the formation volume factor of the oil phase, α_c is the volume conversion factor, ϕ is the porosity. We consider a slightly compressible flow, then the formation volume factor B_o is defined as:

$$B_o = \frac{B_o^0}{1 + c(p - p^0)} \tag{3.11}$$

where c is the compressibility factor, B_o^0 is a reference formation volume factor and p^0 is a reference pressure. For a slightly compressible flow, we assume $1+c(p-p^0) \approx 1$, and that the porosity is constant [1]. Consequently, Equation 3.10 becomes:

$$\frac{\partial}{\partial x}\left(\beta_c k_x \frac{\partial p}{\partial x}\right) + \frac{\partial}{\partial y}\left(\beta_c k_y \frac{\partial p}{\partial y}\right) + \frac{\partial}{\partial z}\left(\beta_c k_z \frac{\partial p}{\partial z}\right) + B^0 \mu q_{sc} = \frac{\mu\phi c}{\alpha_c}\frac{\partial p}{\partial t} \tag{3.12}$$

In the homogeneous isotropic case, we have:

$$k_x = k_y = k_z \tag{3.13}$$

Based on Equation 3.13, and by dividing both sides of the Equation 3.12 by $\beta_c k$, Equation 3.12 becomes:

$$\frac{\partial}{\partial x}\left(\frac{\partial p}{\partial x}\right) + \frac{\partial}{\partial y}\left(\frac{\partial p}{\partial y}\right) + \frac{\partial}{\partial z}\left(\frac{\partial p}{\partial z}\right) + \frac{B^0 \mu q_{sc}}{k\beta_c} = \frac{\phi\mu c}{\beta_c \alpha_c k}\frac{\partial p}{\partial t} \tag{3.14}$$

As mentioned previously, an oil reservoir simulator uses division into a grid. Equation 3.12 is then discretized in space and time expressed over this grid to produce a linear system of equations. We use the CG method which is one of the popular iterative solvers widely used in oil reservoir simulation to solve the equations and find the unknown values which are the pressure for each element of the oil reservoir grid.

4. 3-D Oil-Phase Reservoir Numerical Model. Time discretization of Equation 3.12 gives

$$\frac{p^{n+1}_{i-1jk} - 2p^{n+1}_{ijk} + p^{n+1}_{i+1jk}}{\Delta x^2} + \frac{p^{n+1}_{ij-1k} - 2p^{n+1}_{ijk} + p^{n+1}_{ij+1k}}{\Delta y^2} + \frac{p^{n+1}_{ijk-1} - 2p^{n+1}_{ijk} + p^{n+1}_{ijk+1}}{\Delta z^2} + \frac{B^0 \mu q_{sc}}{k\beta_c} = \frac{\phi\mu c}{\beta_c \alpha_c k}\left(\frac{p^{n+1}_{ijk} - p^{n}_{ijk}}{\Delta t}\right) \qquad (4.1)$$

Equation 4.1 can be represented as a linear system of equations of the form

$$Ax = b$$

where A is a matrix which reflects the coefficients in Equation 4.1. The vector variable x represents the unknown pressures (one unknown per grid cell) and b is a constant which is computed based on the pressures calculated at the previous time step. Equation 4.2 provides the solution to a pressure per grid cell. Figure 4.1 shows a numerical stencil for a 3-D block:

$$x_{ijk} = p_{ijk} \quad i = 1 \ldots, N_x, j = 1, \ldots, N_y, k = 1, \ldots, N_z \qquad (4.2)$$

where N_x is the number of cells of the oil reservoir in the x direction, N_y is the number of cells of the reservoir in the y direction and N_z is the number of cells of the reservoir in the z direction. From a programming model point of view, and for the rest of the paper we will use A, x, and b as notations to represent the linear system of equations generated in a time step.

Figure 4.2 shows a computational mesh for a discretized 3-D oil reservoir of size N ($N = N_xN_yN_z$), which is the size of the generated matrix A. The mesh reveals the way the unknown vector is composed. By numbering the unknowns, the resulting linear system of equations following a discretization is represented by a heptadiagonal structured sparse matrix as shown in Figure 4.3.

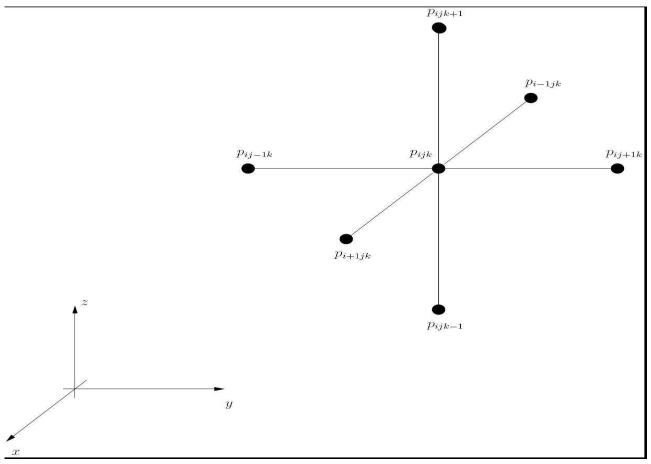


Fig. 4.1: A numerical stencil for a 3-D oil reservoir block.

5. Programming Model.

5.1. Sequential CG Algorithm. As shown in Figure 5.1, the CG method starts with a random initial guess of the solution x_0 (step 1). Then, it proceeds by generating vector sequences of iterates (i.e., successive approximations to the solution (step 10)), residuals corresponding to the iterates (step 11), and search directions used in updating the iterates and residuals (step 14). Although the length of these sequences can become large, only a small number of vectors need to be kept in memory. In every iteration of the method, two inner products (in steps 9 and 13) are performed in order to compute update scalars (steps 9 and 13) that are defined to make the sequences satisfy certain orthogonality conditions. On a symmetric positive definite linear system these conditions imply that the distance to the true solution is minimized in some norm (step 12).

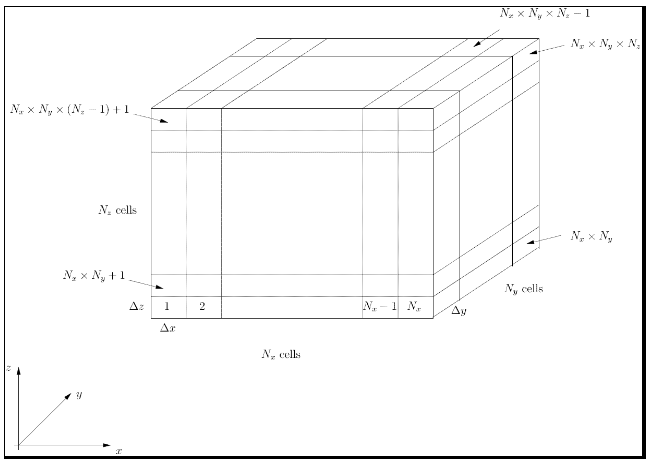


Fig. 4.2: Computational mesh for discretized 3-D oil reservoir.

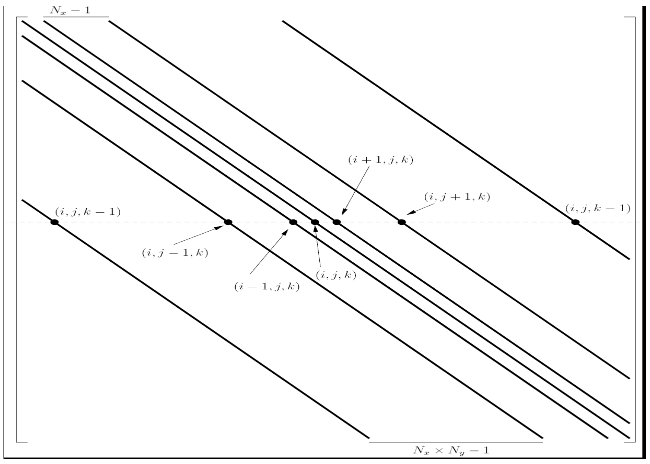


Fig. 4.3: Heptadiagonal coefficient matrix formed from a discretized 3-D oil reservoir.

5.2. Parallel Computation Approach. An oil reservoir simulator along with its CG method solver can be parallelized in different ways. Time parallelism, data parallelism and functional parallelism are common approaches for parallelizing applications. In this work, we choose a mix of functional and data parallelism to parallelize the simulator. The simulator is intuitively divided into 2 functional parts: the simulator itself which includes the numerical discretized part and the CG method iterative solver for solving the numerical system of equations. The CG method is used at every time step. Every time step of the oil reservoir simulator produces a linear system of equations.

We chose the master-worker model [28] as an underlying mechanism of parallelization. The simulator itself is executed sequentially by the master processor. The master processor computes various coefficients and parameters and distributes the matrix relative to the resultant linear system of equations to the available processors who will start the parallel processing of finding a solution. The master processor gathers the output from the different processors involved in the computation which forms the global solution. In each iteration of the CG method, each computational component can be parallelized to compute part of the output values: α_k, x_{k+1}, r_{k+1}, β_k, and p_{k+1}. To achieve the load balancing the number of non-zero values is distributed equally over the number of processors in a greedy-based approach.

5.3. Parallel Implementation. The main goal here is to divide the number of operations of the CG method by the number of available processors to increase its performance vis-a-vis its sequential execution. The flow chart presented in Figure 5.1 presents 2 types of divisible loads: 1) the sparse matrix-vector multiplication

```
1. $x_0 = 0$
2. $r_0 := b - Ax_0$
3. $p_0 := r_0$
4. $k := 0$
5. $Kmax :=$ maximum number of iterations to be done
6. if $k < kmax$ then perform 8 to 16
7. if $k = kmax$ then exit
8. calculate $v = Ap_k$
9. $\alpha_k := \frac{r_k^T r_k}{p_k^T v}$
10. $x_{k+1} := x_k + \alpha_k p_k$
11. $r_{k+1} := r_k - \alpha_k v$
12. if $r_{k+1}$ is sufficiently small then go to 16 end if
13. $\beta_k := \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$
14. $p_{k+1} := r_{k+1} + \beta_k p_k$
15. $k := k + 1$
16. $result = x_{k+1}$
```

Fig. 5.1: CG method sequential algorithm.

(SpMV) presented in step 8 of the flowchart, and 2) the scalar-vector and/or vector-vector operations presented in steps 9, 10, 11, 13, and 14 of the flowchart. However, CG method presents interdependency between its computational elements. In previous work [29], we defined a dependency graph among the different computational parts of the CG as shown in Figure 5.2. This dependency graph gives directions of data flow within one iteration within a processor and among the processors of the system. The graph shows values which are dependent on other values which are connected to and which are higher in the graph representation. For example, α_k is dependent on r_k and p_k.

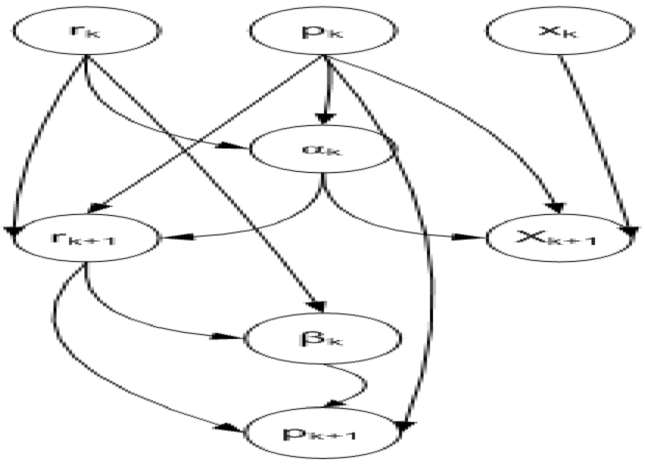


Fig. 5.2: Dependency graph among the different computational elements of the CG method.

For storing the matrix, we use our indexing approach, where the matrix is stored in 2 arrays: a first array which holds the non-zero values and a second array which holds the coordinates of the value in the matrix.

Load balancing is done using a greedy approach by the master processor. The master processor first divides the number of nonzero values in the matrix A by the number of parallel processors to compute the average load per processor. Then the master processor allocates the first n number of rows to the first processor where total number of non-zeros in those n rows are exactly equal to or just more than the calculated average load value. Once the load for a processor in terms of the non-zeros allocated to the processor is calculated, the master processor recalculates the new remaining non-zeros in the matrix by subtracting the number of non-zeros allocated to the processor from the existing value of the remaining non-zeros. Initially all the non-zeros in the matrix A are the remaining non-zeros. Then, the master processor calculates a new average load value (in terms of non-zeros) from the number of remaining non-zeros and the number of remaining processors. The master processor allocates the average number of non-zero elements to the next processor and repeats the same steps till all the non-zero values of the matrix A are allocated to the processors. Since we are using greedy approach for the load distribution purpose and the rows are considered as a unit (fraction of the rows are not given to any process), the method is semi-optimized. Appendix A shows the load balancing algorithm.

Given the interdependency nature of the CG method among its computational steps at each each iteration, the SpMV in step 8 should be distributed in a way to decrease communication cost [30]. We rely on a ring-based approach which allows communications and computations to overlap [11] for the SpMV part in each iteration of the CG. The algorithm works as follows. For the entire local SpMV, every processor needs the whole p vector. Every processor divides its local SpMV into N steps, where N is the number of processors involved in the computation. Initially, every processor has its own part of the vector p. In each step, before starting the local SpMV, a processor sends its own part of the vector p, in a non-blocking communication, to the left neighbor and simultaneously receive part of the vector p from right neighbor forming a ring of communication. The communication takes place in the form of a ring. Figure 5.3 illustrates the starting computational part in each processor. The local SpMV starts on the block number for which the processor has its own chunk of p. The local SpMV is performed using the non-zero elements of the respective blocks. Figure 5.4 shows an example of the computational steps of the processor of rank 0. Appendix B shows the algorithm of the ring-based approach applied to the matrix-vector multiplication step of the CG method.

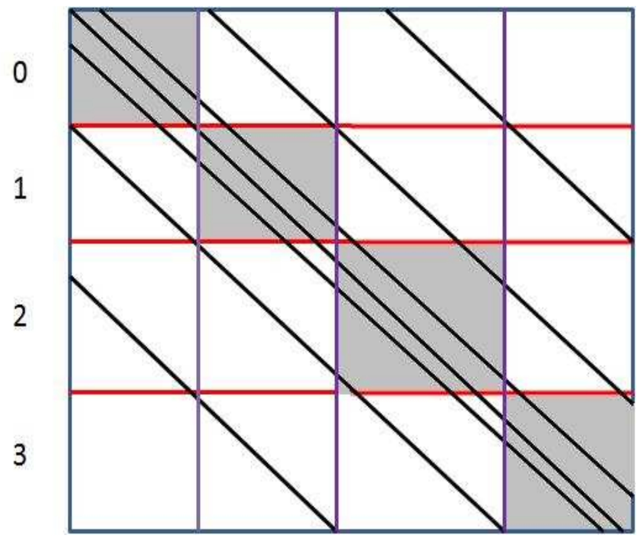


Fig. 5.3: Initialization of Computing at every processor.

6. Evaluation of the Parallel Algorithm. In this section, we evaluate the performance of the ring-based parallel oil-phase reservoir simulator in our experimental environment. We compare the performance of our approach to PETSc-based parallel oil reservoir simulator.

6.1. Experimental Environment. The experiments are conducted on a grid of Xeon Intel Quad Core 5355 machines with 2.66 GHz of CPU. Each machine has a dual CPU. Each core has 4MB of cache, 1GB of memory, 2.66 x 4GFLOPS of peak performance. The machines are connected using InfiniBand (IB) standard. The operating system used on the machines is Red Hat Enterprise Linux Server release 5.2. Message Passing Interface [31] (Open MPI version 1.3.2) library is used for implementing the parallel oil reservoir simulator. We used the mpicc compiler along with gcc version 4.1.2. We used the O3 optimization flag option when compiling the parallel oil reservoir simulator code.

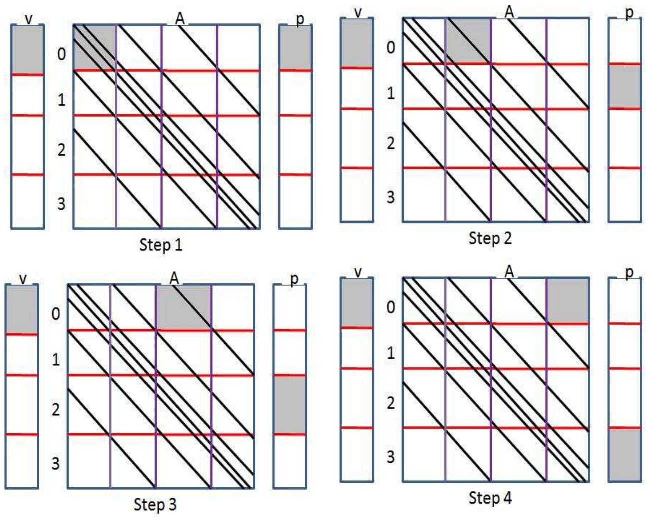


Fig. 5.4: The different computational steps for computing the local SpMV in each processor.

Table 6.1: Experimental Runs

Run	Workload	
	Benchmark Name	Matrix Size(A)
1	S	1400
2	W	7000
3	A	14000
4	B	75000
5	C	150000

6.2. Experiments. The experiments use matrices of different dimensions to assess the performance of the parallel oil reservoir simulator within one step on a single Intel parallel machine and on a grid of Intel parallel machines. The matrix sizes used are as per NAS CG parallel benchmark [26], as shown in Table 6.1. The parallel oil reservoir simulator is measured by horizontally scaling the number of cores up to 128 cores, and vertically scaling the simulation size. The speedup of the parallel parallel oil reservoir versus its sequential execution is measured. We implemented 2 versions of the parallel oil reservoir simulator, one which uses our ring-based approach, and one which uses the PETSc approach in parallelization.

In our experiments, one core acts as a master which distributes the tasks to the other cores that we call workers. The master core runs the simulation, updates and distributes the coefficients; i.e. the matrix, to the workers cores. The *gettimeofday* function is used to compute the elapsed time of the parallel oil reservoir simulator on the master in a single time step. In the sequential execution case, the *gettimeofday* function is used as well to compute the overall run time. In all our experiments, each experiment was run 100 times and the average was computed. The speedup is then measured.

6.3. Performance Evaluation. As discussed previously, our proposed parallel algorithm follows the functional along with data distribution strategy to distribute the oil reservoir simulator computation load among the processors. The simulator itself is run by a master processor, while the parallel CG method is run by a number of parallel processors. In that way, every processor can perform the operations on the data chunk available to it from the master processor. The master processor participates in the computation as well. In devising our parallel algorithm, the numerical representation of the oil reservoir simulator and its CG linear solver were preserved. We worked on functional parallelism, data parallelism and communication strategies to decrease the simulation total execution time. Figure 6.1 shows the speedup performance of the proposed parallel algorithm, which is 42 times faster than the sequential execution of the simulation using 128 processors . It also shows that our parallel implementation scales well with increasing number of processors and large matrix sizes. For instance class C matrix size scales well with increasing number of processors. This is explained by a good overlap between computation and communication for large matrix sizes thanks to a higher number of non-zeros which is allocated to each processor compared to smaller matrix sizes. While our PETSc-based implementation indicates a good speedup of 42.7, as shown in Figure 6.2 for class B matrix size, the speedup performance does not scale with increasing number of processors. The PETSc-based approach has lower scalability compared to our approach with increasing matrix size and increasing number of processors. This is due to the PETSc using asynchronous all-to-all broadcast of the vector p while a local matrix-vector multiplication is taking place. Consequently, the size and the number of vectors exchanged between the processors increase with increasing matrix size and increasing number of processors.

For smaller matrix sizes (classes S and W), our parallelization approach does not scale beyond 8 cores. This is because some processors receive little or no data and therefore the actual computing time can be much shorter than the time spent in communicating the vector p to other processors; i.e., the processors spend the time waiting for the vector p to arrive than computing. Therefore, more time is spent in communicating than computing and consequently the overall execution time of the application will become longer in case the computation is divided further over a larger number of processors. The PETSc-based approach has better performance than our approach for small matrix sizes and small number of processors, where the communication time spent communicating the vector p between the processors is overlapped with the local computing on each processor.

Implementing a ring-based required more design efforts for the code for communicating the vector p than implementing using PETSc approach. Using PETSc, the code calls high level methods and the parallel implementation is done by the underlying library, while in a ring-based approach, the dispatch of the vector p to the next neighbor and the reception of the vector p from the previous neighbor have to be done before the matrix-vector multiplication within each processor as shown in Appendix B.

The greedy approach we use for distributing data ensures load balancing as shown in Figure 6.3. However, Figure 6.3 shows slight discrepancies in load among the processors. This is due to the fact that we do not allow for a partial distribution of a matrix row to the processors. Consequently, some processors may be allocated more non-zero values than others.

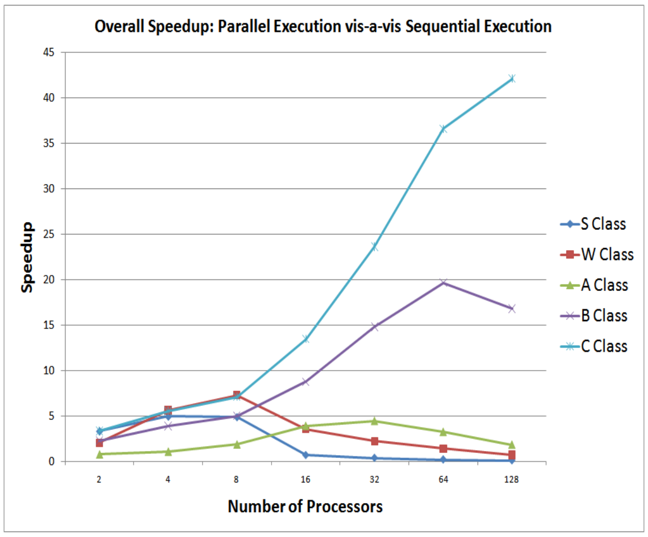


Fig. 6.1: Overall speedup of a parallel 3-D oil-phase reservoir simulator using our approach vis-a-vis its sequential execution.

7. Concluding Remarks. Parallel oil reservoir simulators provide an important computational tool for the oil industry. An oil reservoir simulator involves numerically solving systems of linear equations. The conjugate gradient (CG) method is one of the most popular iterative methods in flow simulation problems. We implemented a parallel oil reservoir simulator using parallel CG. Existing oil reservoir simulators concentrate on data parallelism and load balancing issues and pay less attention to the generated communication cost from

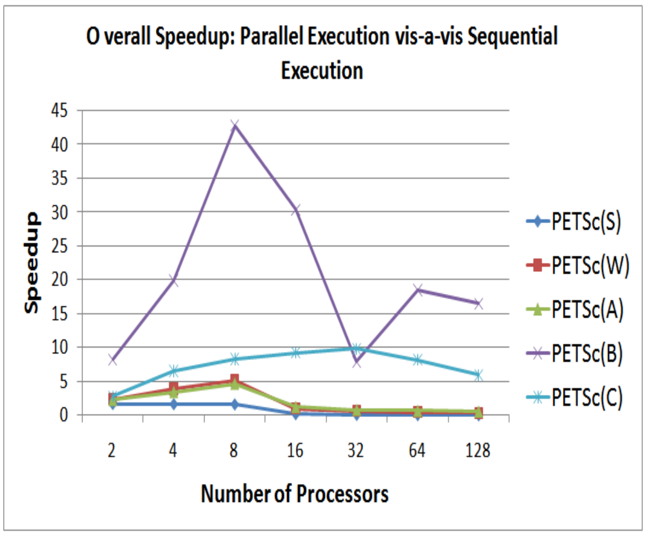


Fig. 6.2: Overall speedup of a parallel 3-D oil-phase reservoir simulator using PETSc vis-a-vis its sequential execution.

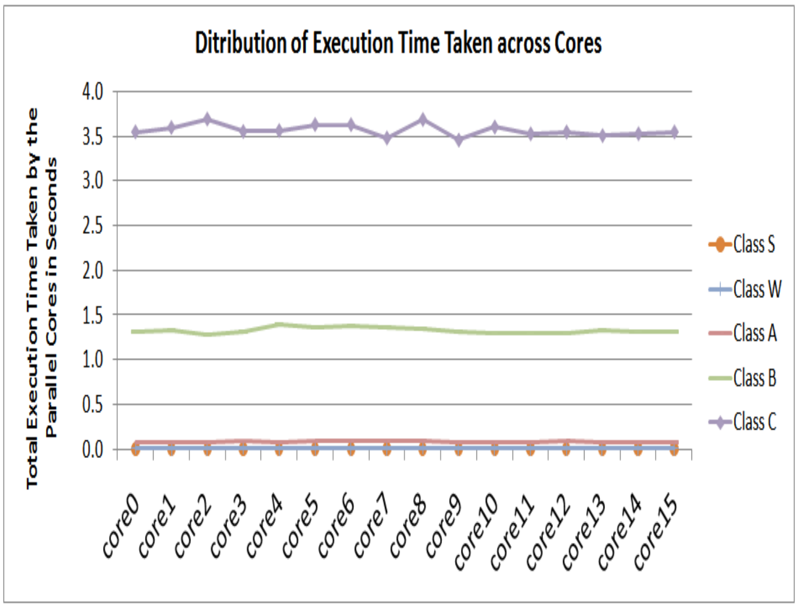


Fig. 6.3: Distribution of execution time taken across the parallel cores.

parallel implementations. In this work, we implemented a ring-based parallel reservoir simulator to reduce communication cost. Our implementation scales well with problem size and with number of processors. A speedup of 42 times was achieved for large problem size vs. a 3-D oil-phase reservoir simulation with sequential execution. We compared our results to the performance of a parallel implementation of the oil reservoir simulator using the Portable Extensible Toolkit for Scientific Computation (PETSc). Our parallel approach scales well with increasing problem size and increasing number of processors compared to our PETSc-based implementation. Our result should be valuable for the oil industry as it should facilitate major savings in engineering effort and result in better oil reservoir management. This work is part of an ongoing project aimed at developing a parallel 3-D multi-phase (oil, gas, and water) reservoir simulator, that applies to heterogeneous and non-isotropic models. The project will develop a model of dynamic distribution of the parallel oil reservoir simulator on a heterogeneous Grid infrastructure of Intel Xeon and IBM Cell processors. Performance evaluations will then be conducted similar to the one performed for our work published in IEEE Transactions on parallel and distributed systems [32].

8. Acknowledgements. The author would like thank the UAE University for supporting this work, which was funded following a national funding competition organized by the UAE National Research Foundation. She would also like to thank Professor Jamal Abou-Kassem from Petroleum Engineering of the UAE University for his help and availability on many useful discussions on oil reservoir simulator. Special thanks to Professor Eyad Abed, the Dean of the Faculty of Information Technology of the UAE University for his inputs and feedback on the paper. Also, thanks to the anonymous reviewers for their feedbacks which have contributed to the paper.

Appendix A. Load Balancing Algorithm.

```
loadBalance( ){
//nnz is the number of non-zero values in the matrix
//nnzLeft is the number of non-zeros left out of the cumulative distributions
int i=0,j=0,k=0;
//The starting row index of the matrix part and the number of rows to be allocated to a process
int *procStartRow, *procCalcRowCount;
//stores number of non zeroes in each row
int *rowDataCount;
//The number of non-zeros allocated to a processor
int *nnzProc;
//Average load (number of non-zeros) to be distributed to each worker
avgLoad = nnz/size;
//Loop over the number of available processors
for(i=0; i ≤ numberOfProcessors-1; i++) {
            //The beginning row index of the next process is equal to the previous process row index
            // added to the load allocated to the previous processes
            processStartRow[i] = processStartRow[i-1] + procCalcRowCount[i-1]
            //Compute the actual load to be allocated to the process
            for (j=0; j ≤ N-1, j++) {
              k = 1;
              nnzProc[i]= nnzProc[i]+rowDataCount[k];
              k = k+1;
              procCalcRowCount[i]=k;
              nnzProc[i]= nnzProc[i]+rowDataCount[k];
              if (nnzProc[i] ≥ avgLoad)
              break;
            }
            nnzLeft=nnzLeft-nnzProc[i];
            int remainingProcessors = numberOfProcessors-1;
            avgLoad = nnzLeft/remainingProcessors;
}
}
```

REFERENCES

[1] Abou-Kassem, J.H., Farouq Ali, S.M., and Islam, M.R., 2006, "Petroleum Reservoir Simulation: A Basic Approach", Gulf Publishing Company, Houston, TX, USA, 480 pp.

[2] J. Aarnes et al, "Towards Reservoir Simulation on Geological Grid Models", 9th European Conference on the Mathematics of Oil Recovery, Cannes, France, September 2004.

[3] Dogru, Ali h., "From Mega-Cell to Giga-Cell Reservoir Simulation", Saudi Aramco Journal of Technology, Spring 2008.

[4] M.R. Hestenes and E. Stiefel, "Methods of conjugate gradients for solving linear systems", J. Research of the National Bureau of STandards, Vol. 49, No. 6, pp. 409-436, 1952.

Appendix B. Ring-Based Distribution Algorithm.

```
parallelMultiplyBlocks(double *vecResult, double **AMatrix,double *p, int *nnzLocalBlock){
//List of parameters of the function parllelMultiplyBlocks:
// vecResult stores the result of the local SpMV by the processor
// AMatrix: Matrix A, stored in a 2D array to represent blocks
// nnzLocalBlocks is the number of local blocks in a local matrix to a processor
// neighbourBack,neighbourNext are ranks of neighbours:
// back is the one to send to ahead and next is the one from whom to receive
// pChunkNumber is the id of the processor which uses the current part of P (the chunk from where the
MVM will start for a processor
// pChunkNumberNext is the ID for processor from whom to receive chunk of P

 int neighbourBack,neighbourNext, pChunkNumber, pChunkNumberNext;

pChunkNumber = processorNumber; // we shall start from blockid = processorNumber.
pChunkNumberNext = (processorNumber + 1) % numberOfProcessors; // we will recveive from next
// neighbour
for(int h = 0;h≤numberOfProcessors-1;h++){// h loops over blocks
 if( h= numberOfProcessors-1 ){
 //Asynchronous send of the part of the vector p that the processor has updated it to the back neighbor
 asynchronousSend( &p[pChunkNumber], neighbourBack);
 //Asynchronous received of the part of the vector p from the next neighbor
 asynchronousReceive( &p[pChunkNumber], neighbourNext);
 }
 //local matrix-vector multiplication on the current block
 for(int i = 0;i≤nnzLocalBlock[pChunkNumber]-1;i++){
        vectorResult=AMatrix*p;
        }
 //wait to receive the chunk of the vector p from the neighbor
 if( h= numberOfProcessors -1){
 Wait(p);
 }
 pChunkNumber = pChunkNumberNext;
 pChunkNumberNext = (pChunkNumberNext+1) % numberOfProcessors;
 }
}
```

[5] Jonathon Richard Shewchuk, "An Introduction to the Conjugate Gradient Method Without the Agonizing Pain", School of Computer Science, Carnegie Mellon University, Edition 1 1/4.

[6] Dianne P. O'Leary, "Parallel Implementation of the Block Conjugate Gradient Algorithm", Parallel Computing, Vol. 5, pp. 127 - 139, 1987.

[7] Dianne P. O'Leary, "The Block Conjugate Gradient Algorithm and Related Methods", Linear Algebra and its Applications, Vol. 29, pp. 293 - 322, 1980.

[8] J. M. Gratien, T. Guignon, J. F. Magras, P. Q. Quandalle, and O. R. Ricois, "Scalability and Load Balancing Problems in Parallel Oil Reservoir Simulation", 10th European Conference on the Mathematics of Oil Recovery, September 2006.

[9] Cao, H., Tchelepi, H. A., Wallis, J. R., and Yardumian, H., 2005, "Parallel Scalable Unstructured CPR-Type Linear Solver for Reservoir Simulation," paper SPE 96809 presented at the 2005 SPE Annual Technical Conference and Exhibition, Dallas, Texas, USA, Oct. 9 - 12, 2005.

[10] Jesper Larsen, Lars Frellesen, John Jansson, Flemming If, Cliff Addison, Andy Sunderland and Tim Oliver, "Parallel Oil reservoir Simulation", Applied Parallel Computing Computations in Physics, Chemistry and Engineering Science Lecture Notes in Computer Science, 1996, Volume 1041/1996, 371-379.

[11] Mark Hoemmen, "Communication-Avoiding Krylov Subspace Mehthods", PhD Disseration, Spring 2010, http://www.cs.berkeley.edu/ mhoemmen/pubs/thesis.pdf

[12] Portable, Extensible Toolkit for Scientific Computation, http://www.mcs.anl.gov/petsc/index.html

[13] The Center for Petroleum and Geosystems Engineering, University of Texas at Austin,"Reservoir Simulation Joint Industry Project", http://www.cpge.utexas.edu/rsjip/

[14] Jason Abate, "Parallel Compositional Reservoir Simulation on a Cluster of PCs", December 1998.

[15] Lois Curfman Mcinnes, and Barry F. Smith, "PETSC 2.0: A Case Study of using MPI to Develop Numerical Software Libraries", Proceeding of the Euro-Par'99 parallel processing: 5th International Euro-Par Conference, 1999.

[16] Lu, P., Shaw, J.S., Eccles, T.K., Mishev, I.D., Usadi, A.K., Beckner, B.L., "Adaptive Parallel Reservoir Simulation," paper IPTC 12199 presented at the International Petroleum Technology Conference, December 2008.

[17] Khashan, S.A., and Ogbe, D.O., and and Jiang, T.M,, 2002, "Development and Optimization of Parallel Code for Large-Scale Petroleum Reservoir Simulation," J. Can. Petrol. Techno., vol. 41, no. 4, 33-37.

[18] Atan, S., Kazemi, H., and Caldwell, D.H., 2006, "Efficient Parallel Computing Using Multiscale Multimesh Reservoir Simulation," paper SPE 103101presented at the 2006 SPE Annual Technical Conference and Exhibition held in San Antonio, Texas, U.S.A., 24-27 September.

[19] Kevin B. Theobald, Gagan Agrawal, Rishi Kumar, Gerd Heber, Guang R. Gao, Paul Stodghill, and Keshav Pingali, "Landing CG on EARTH: A Case Study of Fine-Grained Multithreading on an Evolutionary Path", Proceedings of the ACM/IEEE conference on Supercomputing, pp. 4 - 4, 2000

[20] Fei Chen, Kevin B. Theobald, and Guang R. Gao, "Implementing Parallel Conjugate Gradient on the EARTH Multithreaded Architecture", Sixth IEEE International Conference on Cluster Computing, pp. 459 - 469, 2004.

[21] Piero Lanucara, and Sergio Rovida, "Conjugate Gradients Algorithms: An MPI-OpenMP Implementation on Distributed Shared Memory Systems", First European Workshop on OpenMP, 1999.

[22] P. Kloos, P. Blaise, and F. Mathey, "Open MP and MPI Programming with a CG Algorithm", Proceedings of the European Workshop on OpenMP, 2000.

[23] Marty R. Field, "Optimizing a Parallel Conjugate Gradient Solver", SIAM Journal on Scientific Computing, Vol. 19, issue 1, pp. 27 - 37, 1998.

[24] Lewis and Van de Geijn, "Distributed memory matrix-vector multiplication and conjugate gradient algorithms"', in Proc. Supercomputing'93, Portland, Oregon, pp. 484-492.

[25] John G. Lewis, David G. Payne, "Matrix-vector multiplication and conjugate gradient algorithms on distributed memory computers", in Proc. Scalable High Performance Computing Conference, 1994, pp. 542-550.

[26] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan and S. Weeratunga, "The NAS Parallel Benchmarks", RNR Technical Report RNR-94-007, March 1994.

[27] John R. Fanchi, "Principles of Applied Reservoir Simulation", ISBN 13: 978-0-7506-7933-6, Elsevier, 2006.

[28] Ian Foster, Designing and Building Parallel Programs, Addison- Wesley (ISBN 9780201575941), 1995.

[29] Leila Ismail, "Communication Issues in Parallel Conjugate Gradient Method using a Star-Based Network". 2010 International Conference on Computer Applications and Industrial Electronics (ICCAIE 2010),December 2010.

[30] Leila Ismail, k. Shuaib, "Empirical Study for Communication Cost of Parallel Conjugate Gradient on a Star-Based Network", ams, pp.498-503, In Proceedings of The 2010 Fourth Asia International Conference on Mathematical/Analytical Modeling and Computer Simulation, 2010, May 2010.

[31] William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, William Saphir and Marc Snir, "MPI: The Complete Reference", Vol. 2, ISBN-10:0-262-57123-4, ISBN-13:978-0-262-57123-4.

[32] Leila Ismail, Driss Guerchi, "Performance Evaluation of Convolution on the Cell Broadband Engine Processor," IEEE Transactions on Parallel and Distributed Systems, vol. 22, no. 2, pp. 337-351, Feb. 2011, doi:10.1109/TPDS.2010.70.

Edited by: Dana Petcu and Daniela Zaharie
Received: March 1, 2012
Accepted: April 15, 2012

AIMS AND SCOPE

The area of scalable computing has matured and reached a point where new issues and trends require a professional forum. SCPE will provide this avenue by publishing original refereed papers that address the present as well as the future of parallel and distributed computing. The journal will focus on algorithm development, implementation and execution on real-world parallel architectures, and application of parallel and distributed computing to the solution of real-life problems. Of particular interest are:

Expressiveness:
- high level languages,
- object oriented techniques,
- compiler technology for parallel computing,
- implementation techniques and their efficiency.

System engineering:
- programming environments,
- debugging tools,
- software libraries.

Performance:
- performance measurement: metrics, evaluation, visualization,
- performance improvement: resource allocation and scheduling, I/O, network throughput.

Applications:
- database,
- control systems,
- embedded systems,
- fault tolerance,
- industrial and business,
- real-time,
- scientific computing,
- visualization.

Future:
- limitations of current approaches,
- engineering trends and their consequences,
- novel parallel architectures.

Taking into account the extremely rapid pace of changes in the field SCPE is committed to fast turnaround of papers and a short publication time of accepted papers.

INSTRUCTIONS FOR CONTRIBUTORS

Proposals of Special Issues should be submitted to the editor-in-chief.

The language of the journal is English. SCPE publishes three categories of papers: overview papers, research papers and short communications. Electronic submissions are preferred. Overview papers and short communications should be submitted to the editor-in-chief. Research papers should be submitted to the editor whose research interests match the subject of the paper most closely. The list of editors' research interests can be found at the journal WWW site (`http://www.scpe.org`). Each paper appropriate to the journal will be refereed by a minimum of two referees.

There is no a priori limit on the length of overview papers. Research papers should be limited to approximately 20 pages, while short communications should not exceed 5 pages. A 50–100 word abstract should be included.

Upon acceptance the authors will be asked to transfer copyright of the article to the publisher. The authors will be required to prepare the text in LaTeX 2$_\varepsilon$ using the journal document class file (based on the SIAM's `siamltex.clo` document class, available at the journal WWW site). Figures must be prepared in encapsulated PostScript and appropriately incorporated into the text. The bibliography should be formatted using the SIAM convention. Detailed instructions for the Authors are available on the SCPE WWW site at `http://www.scpe.org`.

Contributions are accepted for review on the understanding that the same work has not been published and that it is not being considered for publication elsewhere. Technical reports can be submitted. Substantially revised versions of papers published in not easily accessible conference proceedings can also be submitted. The editor-in-chief should be notified at the time of submission and the author is responsible for obtaining the necessary copyright releases for all copyrighted material.