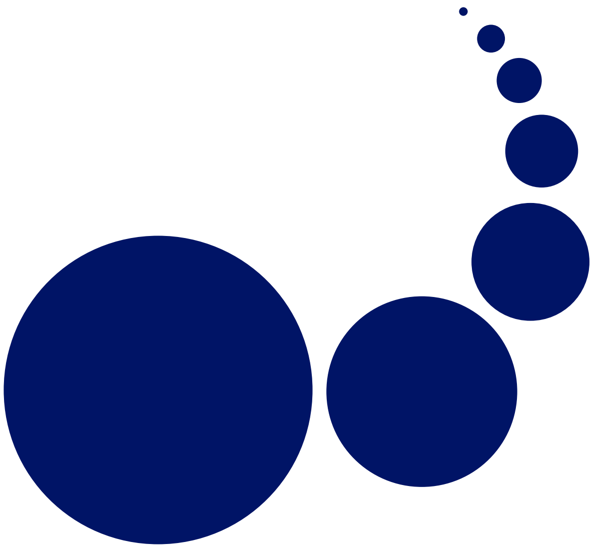
SCALABLE COMPUTING
Practice and Experience



Volume 13, Number 2, June 2012

ISSN 1895-1767

UNIVERSITATEA DE VEST DIN TIMISOARA

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

Scalable Computing: Practice and Experience

Volume 13, Number 2, June 2012

TABLE OF CONTENTS

Scalable Computing: Practice and Experience
Volume 13, Number 2, pp. 100–119. http://www.scpe.org
ISSN 1895-1767
© 2012 SCPE

DEVELOPING DISTRIBUTED SYSTEMS WITH ACTIVE COMPONENTS AND JADEX

LARS BRAUBACH AND ALEXANDER POKAHR*

Abstract. The importance of distributed applications is constantly rising due to technological trends such as the widespread usage of smart phones and the increasing internetworking of all kinds of devices. In addition to classical application scenarios with a rather static structure these trends push forward dynamic settings, in which service providers may continuously vanish and newly appear. In this paper categories of distributed applications are identified and analyzed with respect to their most important development challenges. In order to tackle these problems already on a conceptual level the active component paradigm is proposed, bringing together ideas from agents, services and components using a common conceptual perspective. Besides conceptual foundations of active components also a programming model and an implemented infrastructure are presented. It is highlighted how active components help addressing the initially posed challenges by presenting several real world example applications.

1. Introduction. Technological trends like widespread usage of smart phones and increased internetworking of all kinds of devices lead to new application areas for distributed systems, thus reinforcing and increasing the challenges for their design and implementation. On the one hand, developers can choose from a vast amount of existing technologies, frameworks, patterns, etc. for tackling any challenge that they may face during the development of a complex distributed application. Nonetheless most concrete solutions only address a small set of challenges. Thus for most applications, combinations of different solutions are required, causing a laborious and error-prone process of analyzing, selecting and interating different solution approaches.

On the other hand, a software paradigm represents a holistic solution approach for a more or less generic class of software applications. A paradigm represents a specific worldview for software development and thus defines conceptual entities and their interaction means. It supports developers by constraining their design choices to the intended worldview. When a paradigm fits to the application problem, it allows addressing all challenges using a common conceptual framework, thus effectively reducing the need for the expensive integration and testing of isolated solutions.

The contributions of this paper are as follows. Recurring challenges for the development of todays complex distributed systems are *identified* and existing paradigms, such as object or service orientation, are *analyzed* in which way they support addressing these challenges. As a consequence of the analysis, the *active components* paradigm is proposed as a unification of the strengths of objects, components, services, and agents. The proposed paradigm is concretized on the one hand by a *programming model*, allowing to develop active components systems using XML and Java, and on the other hand by a *middleware infrastructure*, that achieves distribution transparency and provides useful development tools.

The next section presents classes of distributed applications and challenges for developing systems of these classes. Thereafter, the new active components approach is introduced in Section 3. In Section 4 the programming model for active components is introduced and in Section 5 the Jadex platform as active components runtime infrastructure is described. To illustrate the practicality of the approach, several real world example applications are presented in Section 6. Section 7 discusses related work and Section 8 concludes the paper.

2. Challenges of Distributed Applications. The purpose of this paper is conceiving a unified paradigm for developing complex distributed systems. To investigate general advantages and limitations of existing development paradigms for distributed systems, several different classes of distributed applications and their main challenges are discussed in the following. Such challenges arise from different areas and can be broadly categorized into typical *software engineering* challenges for standard applications and new aspects, summarized in this paper as *distribution*, *concurrency,* and *non-functional properties* (cf. also [25]). In Fig. 2.1 theses application classes as well as their relationship to the introduced criteria of software engineering, concurrency, distribution and non-functional aspects are shown. The classes are not meant to be exhaustive, but help illustrating the diversity of scenarios and their characteristics.

Software Engineering: In the past, one primary focus of software development was laid on *single computer systems* in order to deliver typical desktop applications such as office or entertainment programs. Challenges of these applications mainly concern the functional dimension, i.e. how the overall application requirements can be decomposed into software entities in a way that good software engineering principles such as modular design, extensibility, maintainability etc. are preserved.

*Distributed Systems Group, University of Hamburg, {braubach, pokahr}@informatik.uni-hamburg.de

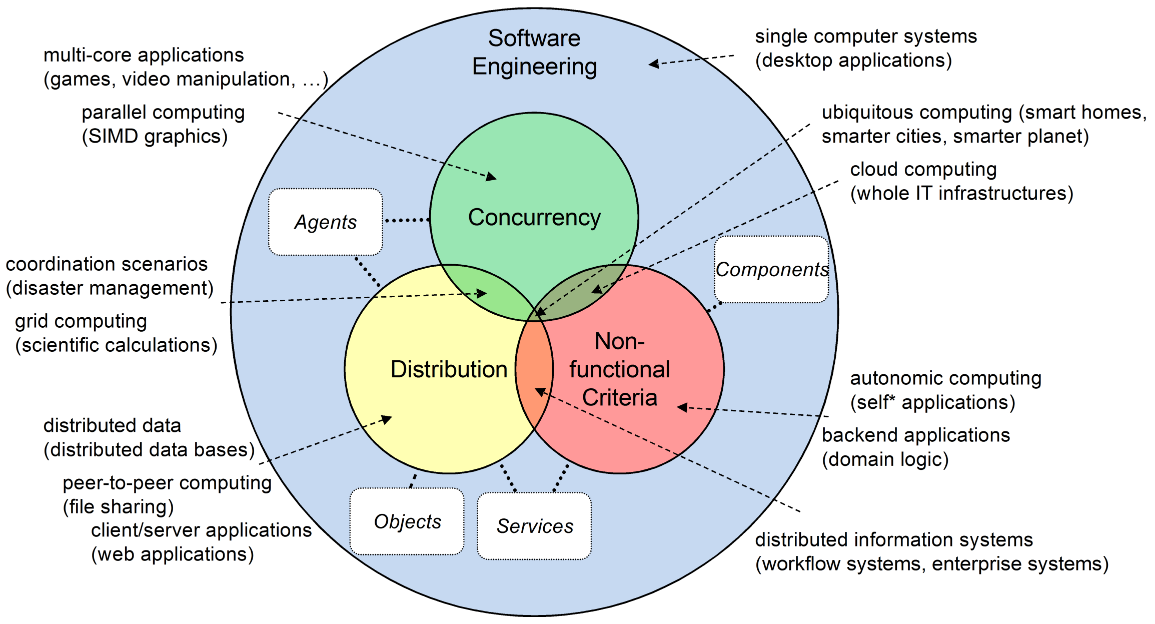

Software Engineering
single computer systems (desktop applications)
multi-core applications (games, video manipulation, ...)
parallel computing (SIMD graphics)
ubiquitous computing (smart homes, smarter cities, smarter planet)
cloud computing (whole IT infrastructures)
Agents
Concurrency
Components
coordination scenarios (disaster management)
grid computing (scientific calculations)
Distribution
Non-functional Criteria
autonomic computing (self* applications)
backend applications (domain logic)
distributed data (distributed data bases)
peer-to-peer computing (file sharing)
client/server applications (web applications)
Objects
Services
distributed information systems (workflow systems, enterprise systems)

Fig. 2.1: Applications and paradigms for distributed systems

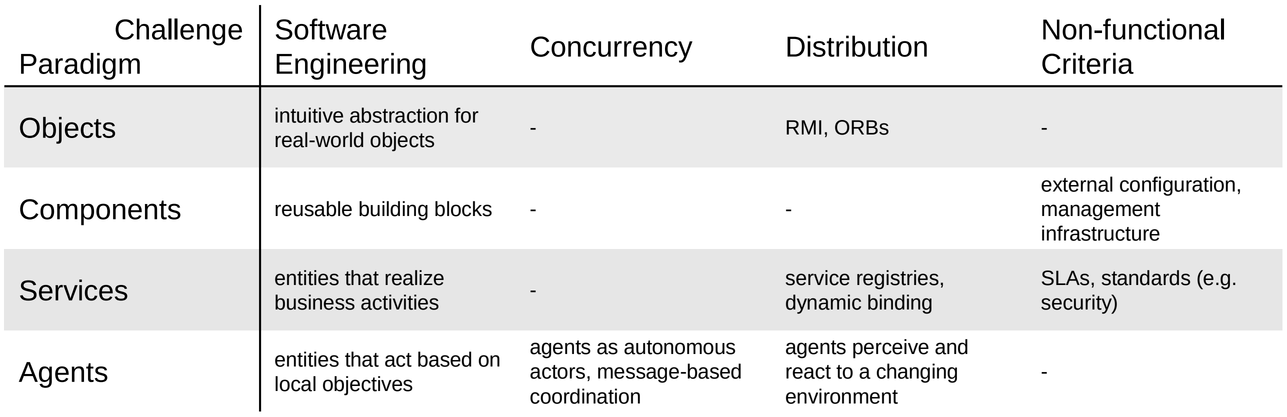
Concurrency: In case of resource hungry applications with a need for extraordinary computational power, concurrency is a promising solution path that is also pushed forward by hardware advances like multi-core processors and graphic cards with parallel processing capabilities. Corresponding *multi-core* and *parallel computing application* classes include games and video manipulation tools. Challenges of concurrency mainly concern preservation of state consistency, dead- and livelock avoidance as well as prevention of race condition dependent behavior.

Distribution: Different classes of naturally distributed applications exist depending on whether data, users or computation are distributed. Example application classes include *client/server* as well as *peer-to-peer computing* applications. Challenges of distribution are manifold. One central theme always is distribution transparency in order to hide complexities of the underlying dispersed system structure. Other topics are openness for future extensions as well as interoperability that is often hindered by heterogeneous infrastructure components. In addition, today's application scenarios are getting more and more dynamic with a flexible set of interacting components.

Non-functional Criteria: Application classes requiring especially non-functional characteristics are e.g. centralized *backend applications* as well as *autonomic computing* systems. The first category typically has to guarantee secure, robust and scalable business operation, while the latter is concerned with providing self-* properties like self-configuration and self-healing. Non-functional characteristics are particularly demanding challenges, because they are often cross-cutting concerns affecting various components of a system. Hence, they cannot be built into one central place but abilities are needed to configure a system according to non-functional criteria.

Combined Challenges: Today more and more new application classes arise that exhibit increased complexity by concerning more than one fundamental challenge. *Coordination scenarios* like disaster management or *grid computing* applications like scientific calculations are examples for categories related to concurrency and distribution. *Cloud computing* subsumes a category of applications similar to grid computing but fostering a more centralized approach for the user. Additionally, in cloud computing non-functional aspects like service level agreements and accountability play an important role. *Distributed information systems* are an example class containing e.g. workflow management software, concerned with distribution and non-functional aspects. Finally, categories like *ubiquitous computing* are extraordinary difficult to realize due to substantial connections to all three challenges.

In this paper *object*, *component*, *service* and *agent orientation* are further discussed as successful paradigms for the construction of real world distributed applications. Fig. 2.2 highlights which challenges a paradigm conceptually supports. Object orientation has been conceived for typical desktop applications to mimic real



Challenge / Paradigm	Software Engineering	Concurrency	Distribution	Non-functional Criteria
Objects	intuitive abstraction for real-world objects	-	RMI, ORBs	-
Components	reusable building blocks	-	-	external configuration, management infrastructure
Services	entities that realize business activities	-	service registries, dynamic binding	SLAs, standards (e.g. security)
Agents	entities that act based on local objectives	agents as autonomous actors, message-based coordination	agents perceive and react to a changing environment	-

Fig. 2.2: Contributions of paradigms

world scenarios using objects (and interfaces) as primary concept and has been supplemented with remote method invocation (RMI) to transfer the programming model to distributed systems. Component orientation extends object oriented ideas by introducing self-contained business entities with clear-cut definitions of what they offer and provide for increased modularity and reusability. Furthermore, component models often allow non-functional aspects being configured from the outside of a component. The service oriented architecture (SOA) attempts an integration of the business and technical perspectives. Here, workflows represent business processes and invoke services for realizing activity behavior. In concert with SOA many web service standards have emerged contributing to the interoperability of such systems. In contrast, agent orientation is a paradigm that proposes agents as main conceptual abstractions for autonomously operating entities with full control about state and execution. Using agents especially intelligent behavior control and coordination involving multiple actors can be tackled.

Yet, none of the introduced paradigms is capable of supporting concurrency, distribution and non-functional aspects at once, leading to difficulties when applications should be realized that stem from intersection categories (cf. Fig. 2.1). In order to alleviate these problems already on a conceptual level, in the following section the active component paradigm is proposed as a unification of the analyzed paradigms.

3. Active Components Paradigm. For addressing all challendes of distributed systems in a unified way, the active component paradigm brings together agents, services and components in order to build a worldview that is able to naturally map all existing distributed system classes to a unified conceptual representation [24]. Recently, with the service component architecture (SCA) [20] a new software engineering approach has been proposed by several major industry vendors including IBM, Oracle and TIBCO. SCA combines in a natural way the service oriented architecture (SOA) with component orientation by introducing SCA components communicating via services. Active components build on SCA and extend it in the direction of sofware agents. The general idea is to transform passive SCA components into autonomously acting service providers and consumers in order to better reflect real world scenarios which are composed of various active stakeholders. In Fig. 3.1 an overview of the synthesis of SCA and agents to active components is shown. In the following subsections the implications of this synthesis regarding structure, behavior and composition are explained.

3.1. Active Component Structure. In Fig. 3.1 (right hand side) the structure of an active component is depicted. It yields from conceptually merging an agent with an SCA component (shown at the left hand side). An agent is considered here as an autonomous entity that is perceiving its environment using sensors and can influence it by its effectors. The behavior of the agent depends on its internal reasoning capabilities ranging from rather simple reflex to intelligent goal-directed decision procedures. The underlying reasoning mechanism of an agent is described as an agent architecture and determines also the way an agent is programmed. On the other side an SCA component is a passive entity that has clearly defined dependencies with its environment. Similar to other component models these dependencies are described using required and provided services, i.e. services that a component needs to consume from other components for its functioning and services that it provides to others. Furthermore, the SCA component model is hierarchical meaning that a component can be composed of an arbitrary number of subcomponents. Connections between subcomponents and a parent component are

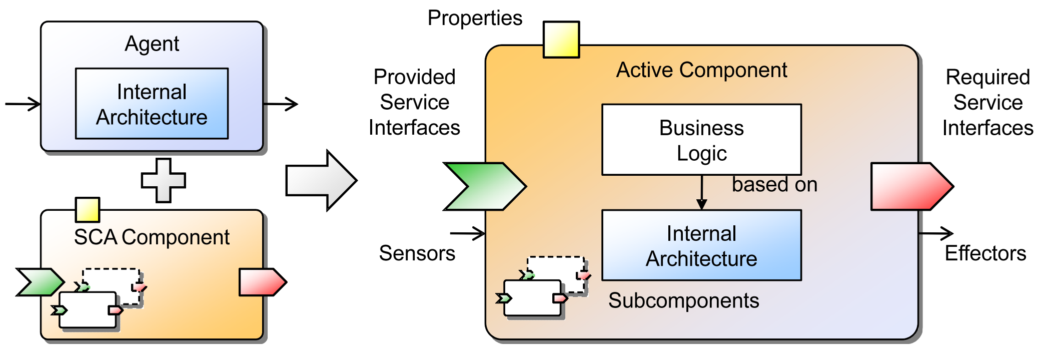


Fig. 3.1: Active component structure

established by service relationships, i.e. connection their required and provided service ports. Configuration of SCA components is done using so called properties, which allow values being provided at startup of components for predefined component attributes. The synthesis of both conceptual approaches is done by keeping all of the aforementioned key characteristics of agents and SCA components. On the one hand, from an agent-oriented point of view the new SCA properties lead to enhanced software engineering capabilities as hierarchical agent composition and service based interactions become possible. On the other hand, from an SCA perspective internal agent architectures enhance the way how component functionality can be described and allow reactive as well as proactive behavior.

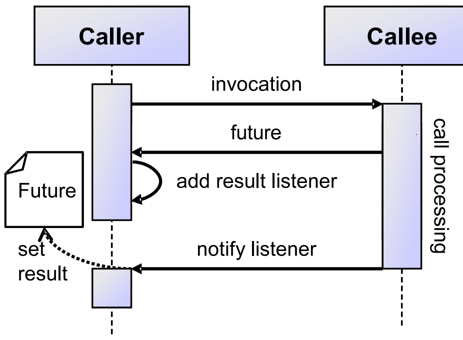
3.2. Behavior. The behavior specification of an active component consists of two parts: service and component functionalities. Services consist of a service interface and a service implementation. The service implementation contains the business logic for realizing the semantics of the service interface specification. In addition, a component may expose further reactive and proactive behavior in terms of its internal behavior definition, e.g. it might want to react to specific messages or pursue some individual goals.

Due to these two kinds of behavior and their possible semantic interferences the service call semantics have to be clearly defined. In contrast to normal SCA components or SOA services, which are purely service providers, agents have an increased degree of autonomy and may want to postpone or completely refuse executing a service call at a specific moment in time, e.g. if other calls of higher priority have arrived or all resources are needed to execute the internal behavior. Thus, active components have to establish a balance between the commonly used service provider model of SCA and SOA and the enhanced agent action model. This is achieved by assuming that in default cases service invocations work as expected and the active component will serve them in the same way as a normal component. If advanced reasoning about service calls is necessary these calls can be intercepted before execution and the active component can trigger some internal architecture dependent deliberation mechanism. For example a belief desire intention (BDI) agent could trigger a specific goal to decide about the service execution.

To allow this kind service call reasoning service processing follows a completely asynchronous invocation scheme based on futures. The service client accesses a method of the provided service interface and synchronously gets back a future representing a placeholder for the asynchronous result. In addition, a service action is created for the call at the receivers side and executed on the service's component as soon as the interpreter selects that action. The result of this computation is subsequently placed in the future and the client is notified that the result is available via a callback.

In the business logic of an agent, i.e. in a service implementation or in its internal behavior, often required services need to be invoked. The execution model assures that operations on required services are appropriately routed to available service providers (i.e. other active components) according to a corresponding binding. The mechanisms for specifying and managing such bindings are part of the active component composition as described next.

3.3. Composition. One advantage of components compared to agents is the software engineering perspective of components with clear-cut interfaces and explicit usage dependencies. In purely message-based agent systems, the supported interactions are usually not visible to the outside and thus have to be documented separately. The active components model supports the declaration of provided and required services and advo-



```
IFuture<String> fut = callee.method(arg1, arg2);
fut.addResultListener(new IResultListener<String>() {
  public void resultAvailable(String res) {
    System.out.println( "System.out.println( "Result: "+res)" );
  }
  public void exceptionOccurred(Exception e) {
    System.out.println( "System.out.println( "Exception: "+e)" );
});
```

Fig. 4.1: Asynchronous method invocation with future return value

cates using this well-defined interaction model as it directly offers a descriptive representation of the intended software architecture. Only for complex interactions, such as flexible negotiation protocols, which do not map well to service-based interactions, a more complicated and error-prone message-based interaction needs to be employed.

The composition model of active components thus augments the existing coupling techniques in agent systems (e.g. using a yellow page service or a broker) and can make use of the explicit service definitions. For each required service of a component, the developer needs to answer the question, how to obtain a matching provided service of a possibly different component. This question can be answered at design or deployment time using a hard-wiring of components in corresponding component or deployment descriptors. Yet, many real world scenarios represent open systems, where service providers enter and leave the system dynamically at runtime [15]. Therefore, the active components approach supports besides a static wiring (called *instance* binding) also a *creation* and a *search* binding (cf. [24]). The *search* binding facilities simplified specification and dynamic composition as the system will search at runtime for components that provide a service matching the required service. The creation binding is useful as a fallback to increase system robustness, e.g. when some important service becomes unavailable.

The active components paradigm introduced in the last sections allows a conceptual view of a distributed system as a dynamic composition of autonomously executing entities with clearly defined interfaces. Yet, the conceptual view leaves open many questions with regards to how the behavior of a component is realized or how the interaction between components looks like. These questions are answered by a concrete active components programming model introduced next.

4. Programming Model. In this section the general concepts of active components, as presented before, will be further refined to a concrete programming approach. The approach itself is similar to the SCA programming model with the following major exceptions. First, the programming model of active components is inherently asynchronous, which is also directly reflected in the way service interfaces are specified and services have to be implemented.[1] Second, components may expose their own behavior in addition to providing external services. For this reason the programming concepts for components heavily depend on their concrete internal architectures. Third, as bindings between components can be configured to be dynamic, programming component compositions introduces new means for declarative search specifications. In the following, a short introduction to the underlying asynchronous programming model with future based return values is given. Thereafter, the key aspects from the last section - structure, behavior and composition - will be revisited on the programming level.

4.1. Asynchronous Programming with Futures. The widely used synchronous message based invocation scheme well known from object-oriented programming is easy to understand and employ. It fits to the fundamental idea of the imperative programming paradigm considering programs as a linear sequence of actions. Actions are processed one by one and the next action is begun only after completion of its predecessor. In case of distributed applications this style of programming leads to severe problems as it means that an action possibly has to wait for completion of a called remote action e.g.

[1]This does not mean that SCA does not support asynchronous invocations at all. In SCA the callback pattern is used to pass callback objects as parameters from the caller to the callee. The callee can use the interface of the callback object to invoke its remote methods. This approach leads to interface definitions that look synchronous but in fact are not.

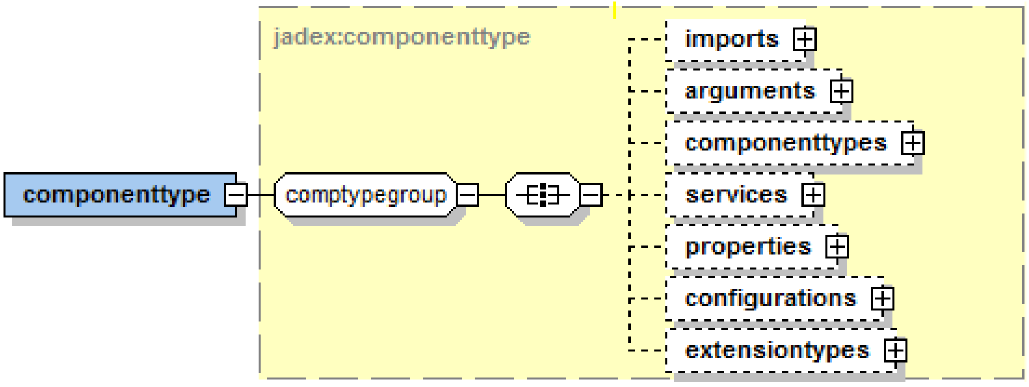


Fig. 4.2: Active component structure specification

via remote procedure call or remote method invocation. Hence, processing of the caller has to be blocked until the result of the callee arrives. Besides being inefficient this invocation scheme is inherently deadlock prone because invocation cycles between callers and callees can easily occur, e.g. if the callee needs a functionality of the caller and invokes one of its methods it cannot be served as the caller is still blocked. Such technical deadlocks can be avoided when an asynchronous invocation scheme is employed. In this case the caller is not blocked after issuing a call and can continue processing other tasks. In practice, asynchronous programming has become common with several important technologies like AJAX in the context of HTTP processing and the GoogleAppEngine for realizing cloud applications.

Futures [29] have been developed as fundamental programming concept for asynchronous systems and represents a holder for the future result of an initiated processing. In case of an asynchronous call with future return value, the callee immediately returns the future object to the caller. The caller can use the future to check if the result has been provided and read the real result value. Typically, futures provide some form of a blocking get method that the caller can invoke to become suspended until the result has been made available. It has to be noted that this *wait-by-necessity* mechanism again opens up the possibilities for deadlocks and should be avoided. Instead, a result listener should be used that is notified in the moment the result value arrives.

In Figure 4.1 the concept of an asynchronous call with future result value is visualized and also the corresponding Java code is shown. It can be seen that the caller invokes a method on the callee, which starts processing the call. In the example code (line 1) two arguments (called arg1, arg2) are passed to the callee. As result type a future is defined (*IFuture* represents the interface for futures). Java generics are used to specify the type of the real return value of the future (here String). The callee returns a future to the caller as soon as possible and afterwards may continue processing the request. After the future object has been received by the caller, it adds a result listener to it (line 2) and may or may not continue processing other tasks. The code (lines 2-8) highlights the result listener (*IResultListener*) interface and methods. It contains two obligatory methods named *resultAvailable()* and *exceptionOccurred()*, which are invoked exclusively. The first method is invoked if the call could be processed normally, otherwise the latter one is used to signal the exception that was thrown. Discriminating between both allows for keeping the normal Java method execution semantics, i.e. asynchronous methods can use exceptions to inform the caller about execution problems. After the callee has finished, it will provide the result to the future, which subsequently notifies all registered result listeners at the caller side. In consequence, either the result value (line 4) or the exception (line 7) is printed out to the console by the example listener.

4.2. Component Structure Specification. Active components exhibit a common black box view of properties shown in Figure 4.2.[2] Using these properties a specific component type can be specified from which component instances can be created at runtime (similar to the relation of a Java class and its instances). To foster a general understanding of the component specification first the meaning of these properties will be sketched.

[2]It has to be noted that specification of active components can be done in different formats including XML (following the XML scheme of Figure 4.2) and also Java annotations. The component type, e.g. BPMN workflow or BDI agent, determines the way in which the properties need to be defined.

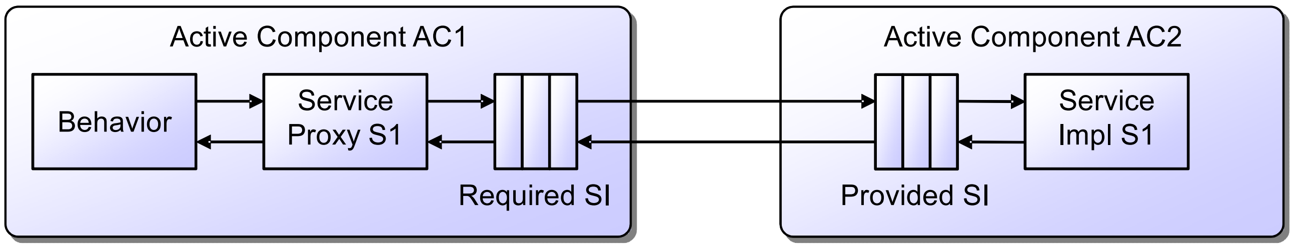


Fig. 4.3: Service interceptors

- *Imports* can be used in the same way as in Java classes to include resources like Java classes and packages that are used in context of the file.
- The *Arguments* section contains both, argument and result types. It can be stated which arguments can be fed into the component at start up and which results are provided by component after termination. For an argument and result, a name, implementation class and default value can be provided. The explicit definition of arguments and result types as part of the public component structure allows for treating components also in a functional way, i.e. one can consider them as a function performing operations on input data and finally producing some output data. This fits well to e.g. workflow based applications in which subworkflows are often invoked with functional semantics.
- In the *Component types* part the types of subcomponents can be defined with a local name and a filename that points to the referenced model. Having local names for subcomponent types facilitates the definition of component instances at other places in the same file.
- The *Services* section contains a definition of the provided and required service types of a component. Details will be presented in the service specification section below.
- *Properties* represent optional settings of a component.
- *Configurations* allow for specifying different component setups that can be used at startup of a component. A configuration is defined with a name and most importantly can be employed to provide composition information about subcomponents and their bindings. At startup of a component the configuration name is used to choose among its predefined configurations, e.g. a test configuration with mock subcomponents vs. an operational setting.

4.3. Service Invocations. Service invocations between active components need to cope with the inherent system concurrency. Each active component may potentially expose active behavior and thus executes proactive behavior on its own thread of control. In order to avoid concurrent access to the state of a component by different components that invoke services at the same time, a general protection mechanism between the caller and callee component is established. This protection mechanism is in charge of decoupling incoming calls from the caller thread and execute them on the callee thread. After the result has been produced the control is transferred back to the caller thread. In this way each component is executed on its own thread only and all data access is linearized. To further protect also data that is transmitted between components as parameter or return values of method invocations it has to be ensured that components do not share those objects and modify them concurrently. State corruption can be avoided by giving components exclusively owned objects and only sharing immutable objects. To assure this property, parameter and return values are automatically cloned if they are mutable. Otherwise direct object references can be provided in local method invocations. In this way active components follow the fundamental principles of the actor model [11] considering each active component as independent actor who's behavior and state is independent of other actors [16].

At the implementation side thread and parameter protection are ensured by using an extended variant of the interceptor design pattern [27]. Using interceptors renders the employed mechanisms transparent for service users and providers. The basic invocation scheme is illustrated in Figure 4.3. Given that some behavior in active component *AC1* wants to invoke a service method on a known service with interface *S1*, the call will be catched by the local required service proxy of *AC1*. This service proxy looks to the service user as if it were the original service but in fact only implements the same service interface S1. The required service proxy owns a chain of asynchronous interceptors (*Required SI*) which are subsequently invoked. The last interceptor in this chain performs a (possibly remote) method call to the active component *AC2*, which is hosting the original service implementation of *S1*. Before the call is routed to the implementation, the interceptor chain of the provided

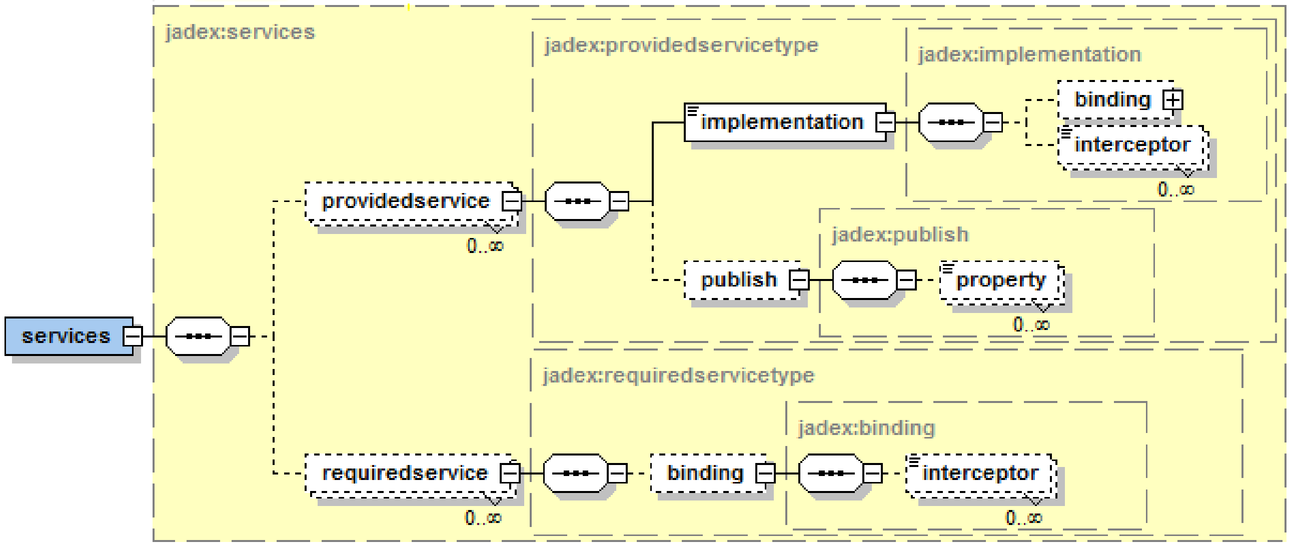


Fig. 4.4: Provided and required service specification

service (*Provided* SI) is executed. Thread decoupling is done here at two points. First, on *AC2* the incoming request is decoupled by an interceptor and finally, at *AC1* the returning invocation is decoupled and routed to the thread of *AC1*. State encapsulation is handled exclusively at the side of the provided interceptor chain. In case of a local call the interceptor clones arguments before the call and the result after the call, whereas in case of a remote call no cloning needs to be performed as the remote method invocation itself has to marshal and unmarshal parameter and return values.

4.4. Service Specification. In Figure 4.4 details of provided and required service specification are depicted. A provided service is defined by using its *interface type* as well as an obligatory *implementation* definition and optional further *publishing* options. The service implementation is typically defined via an implementation class that is used by the component to instantiate the service at component startup. Alternatively, a *binding* can be used to delegate service calls to another subcomponent, i.e. the component does not host the implementation itself but forwards calls to another component. Binding details are described in context of required services below. In addition to the service implementation also custom interceptors can be defined. These interceptors represent an extension point that can be used to insert new behavior in the sense of aspect-oriented programming [17], before, after or around specific service calls. Publishing options can be used to provide a service in other technologies facilitating the interoperability of external systems with the active components runtime. Currently, support exists for publishing active component services as WSDL-based or RESTful web services. The publication process can be done either fully automatically or by providing custom mapping information that describe how the published service should look like. More details about service publication can be found in [6].

Required services are specified using basic required service information and binding details. The first refers to the general characteristics of a required service and includes aspects like the local *name*, the *service interface,* as well as the *multiplicity*. The name is used to refer to the required service declaration from behavior code and the interface describes the expected type of the service. Additionally, for a required service the multiplicity property can be used to state if exactly one service or a set of services should be delivered. The second part of the specification contains details about the search characteristics that are used to locate required services. Most importantly the search space can be defined by using a *search scope*, which describes the components that are included in the service search. Currently, several different default scopes are available that range from local scope, considering only a component itself, over application scope including all components of one specific application to platform and global scope. The latter options include all components on one platform and components of all currently connected remote platforms. Many further options to adjust the search to the concrete application demands are available. Examples include the search *dynamics* and service *recovery*. The first aspect determines if the search should be executed on each service access or the results of a former search should be cached. The latter issues a new service search transparently for a service user if the currently used service becomes unavailable for some reason.

4.5. Component Implementation. The implementation of components consists of two parts. The provided service implementations and the component behavior implementation. Both parts are optional to allow defining components that only contain internal behavior and passive components in the sense of traditional components with no own proactive behavior. The implementation of services is kept as simple as possible by sticking to the Java POJO (plain old Java objects) model, i.e. developers create purely domain oriented classes without having to extend or use framework specific classes or interfaces. Active component specifics are included using Java annotations. Especially, annotations are provided to enable dependency injection [10] of the hosting component itself, required services or component arguments to the service implementation.

The implementation of component behavior is dependent on the concrete type of component used. In the following the implementation principles of two exemplary component types are roughly sketched. The first component type is called *micro agents*, which represents a very simple Java based agent architecture and the second type are *BPMN* (business process modeling notation) *workflows.* Micro agents are defined as annotated Java classes. The architecture assumes a simple three-phased execution model of the internal agent behavior. The three phases are initialization, execution and termination and the infrastructure guarantees that a specific method of the micro agent pojo is called when entering each of the phases. Despite the three phases, a micro agent can implement more complex behavior by scheduling actions at later points in time. Furthermore, reactive behavior can be initiated by arriving service calls or incomings messages. BPMN workflows are modeled graphically according to the corresponding standard [22] mainly with events, actions and gateways. The workflow descriptions need to be enriched with implementation details that are added to the model elements. A Java expression language is used to encode parameter values and constraint checks at gateways. Moreover, domain dependent behavior is encoded in extra Java classes that can be bound to specific actions in the process model.

4.6. Example Implementation. To illustrate the implementation of components further, below a cutout of the implementation of a simple chat micro agent is given. It is a peer-to-peer chat variant in which each chat agent offers a chat service. In Figure 4.5, the chat agent (*ChatAgent*), the chat interface (*IChatService*) as well as a cutout of the service implementation (*ChatService*) are shown. It can be seen that the component file (lines 1-15) contains annotations to declare the active component characteristics and a small behavior part contained in the body method. First of all, the *@Agent* annotation (line 1) is used to state the Java class is an active component declaration. It also declares one provided service (line 2) with interface *IChatService* and an implementation class *ChartService.* This means that the agent will automatically create an instance of the implementation class at startup to provide the given service interface. In addition, a required service with name "chatservices" is defined (line 3), which can be used to retrieve all chat services in a network of platforms. To fetch all services instead of one, the multiplicity has been set to true. The binding of the required service is set to dynamic and to global search scope. This ensures that each service request leads to a fresh search and that all available platforms are included into the search. The behavior of the chat agent (lines 5-14) is annotated with *@AgentBody* and very simple in this case. It creates a command (called component step) that is periodically executed by the agent. Each time the command is invoked it searches the users currently online by using the corresponding required services and refreshes the user list in the user interface.

The chat service interface (lines 17-22) contains methods to send a message (line 19), to actively announce a new user state, e.g. user is typing a message (line 20) and to send a file to another user (line 21). Additionally, the service is annotated with a security setting (line 17), which enables unrestricted access to the chat service, i.e. other platforms can find chat service components even when the platform is password protected and normally restricts search and service requests. The implementation of the service (line 24-35) is identified with the *@Service* annotation. It implements the *IChatService* interface and additionally introduces a lifecycle method named start() (lines 26-29) that is called on initialization of the service and creates the user interface. The implementation of the *message()* method just forwards a received message to the user interface, which will show it to the user. It can be seen that the sender of the message (more precisely the component identifier of the caller) can be always obtained directly via a thread local variable that is provided by the framework (line 31) so that no extra parameter is needed.

After this section has clarified the active components programming model using a concrete example, the next section will introduce a runtime infrastructure and development tools for deploying active components systems in distributed environments.

```
@Agent
@ProvidedServices(@ProvidedService(type=IChatService.class,
  implementation=@Implementation(ChatService.class)))
@RequiredServices(@RequiredService(name=chatservices, type=IChatService.class,
  multiple=true, binding=@Binding(dynamic=true, scope=Binding.SCOPE_GLOBAL)) )
public class ChatAgent {
  @AgentBody
  public void body() {
    IComponentStep<Void> step = new IComponentStep<Void>() {
      public IFuture<Void> execute(IInternalAccess agent) {
        getChartPanel().refreshUserList(searchCurrentUsers());
        agent.waitForDelay(delay, this);
      }
    };
    scheduleStep(step);
  }
}

@Security(Security.UNRESTRICTED)
public interface IChatService {
  public IFuture<Void> message(String text);
  public IFuture<Void> status(String status);
  public IFuture<Void> sendFile(String filename, long size, IInputConnection con);
}

@Service
public class ChatService implements IChatService {
  @ServiceStart
  public IFuture<Void> start() {
    // gui init, creates chat panel
  }
  public IFuture<Void> message(String text) {
    chatpanel.addMessage(IComponentIdentifier.CALLER.get(), text);
    return IFuture.DONE;
  }
  ...
}
```

Fig. 4.5: Chat service interface and implementation snippets

5. Platform Architecture and Implementation. The proposed active components paradigm and programming model require a runtime infrastructure for loading and executing component models and for providing discovery and communication facilities for their composition. Therefore, the active components concepts have been realized in the open source Jadex platform.[3] In the following, the basic architecture and its important modules will be described. The *component container* represents the minimal requirement of being able to execute and manage local components and enable their interaction in terms of provided and required services. In a *distributed infrastructure*, interaction between multiple component containers as well es other external systems needs to be supported. Therefore, important middleware features need to be introduced for supporting and simplifying the development of distributed applications using an active components infrastructure. Finally, *runtime tools* are required to foster, e.g., debugging during systems development as well as administration and monitoring of deployed systems.

5.1. Component Container. The main modules of the platform provide the execution context for any active components running on the platform, i.e. they form the component container. Their interdependencies are illustrated in Figure 5.1. All modules contribute to one or both of the component management and messaging functionalities. Both of these functionalities are further explained in the following two sections, followed by some details about the generic approach towards realizing these functionalities.

5.1.1. Component Management. The *Component Management* module is responsible for starting and stopping components. Upon initialization of each component, its provided services are instantiated and made available for searching and invocation. Additionally, the means for binding and invoking required services are

[3]`http://jadex.sourcefourge.net/`

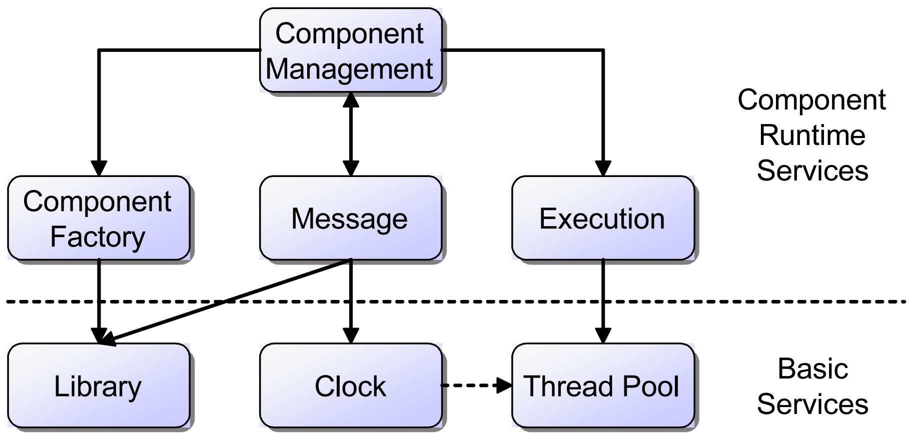


Fig. 5.1: Basic platform services

set up according to the component description or additional configuration options supplied as external start parameters. The component management also serves as an entry point to the platform by providing information about running components on request or in a publish-subscribe fashion.

Component management makes use of the *Component Factory* for loading and instantiating component descriptions. The component factory in turn uses the *Library* module for handling the physical access to component descriptions, e.g. on a local hard drive or in component repositories. Different component factories exist that represent the different component types (cf. Section 4.5). For each component description, thus a component type specific interpreter implementation is initialized. The component management passes the interpreter to the *Execution* module, which is responsible for providing a thread from a *Thread Pool* to the interpreter, whenever the corresponding component should be executed.

5.1.2. Messaging. Each component is assigned a unique id that enables addressing messages to specific components. The *Message* module is responsible for the internal delivery of messages. It further enables tracking of timeouts with the help of the *Clock*, which, in case of an active clock[4], uses a thread from the thread pool. The message module also deals with the marshalling and unmarshalling of message contents, and uses the library module, e.g. for resolving classes for unmarshalling message content into appropriate Java objects.

5.1.3. Container Realization. All of the aforementioned modules are realized as component services. As a result, the platform itself is considered an active component with the platform modules modeled as provided services and their interdependencies being represented as required services. This approach provides a number of technical advantages regarding their implementation. First, the mechanisms for initializing and managing as well as searching and invoking component services are employed for platform services as well, thus reducing the implementation effort for this recurring functionality. Further on, the platform configuration is specified as a component description, such that existing specifications means can be reused and the developer may choose from the available description means like Java or XML, if she wishes to provide a customized platform configuration.

Another advantage is that the execution mechanisms, e.g. for decoupling of asynchronous calls, apply to platform services as well, such that concurrency issues can be avoided in the implementations. Also the dynamic binding of services is of advantage here, as platform services can easily be exchanged in the platform configuration or even at runtime. For example, Jadex supports seamless switching between different clock implementations also when components are currently executing. Last but not least, this approach is easy to realize. Only a simple bootstrapping script is required that loads and instantiates a platform configuration through a predefined component factory and calls the obtained interpreter until the actual execution service is available. As a result, the platform itself is highly configurable and can be adapted to the needs of an application using the same concepts that are also used for application implementation. This is also illustrated in the next section that introduces additional platform services for supporting distributed infrastructures.

[4]Jadex supports different clock types including active clocks for normally timed or dilated execution as well as passive clocks, which are controlled by an additional simulation module, e.g. for as-fast-as-possible execution of simulation scenarios as described in [26].

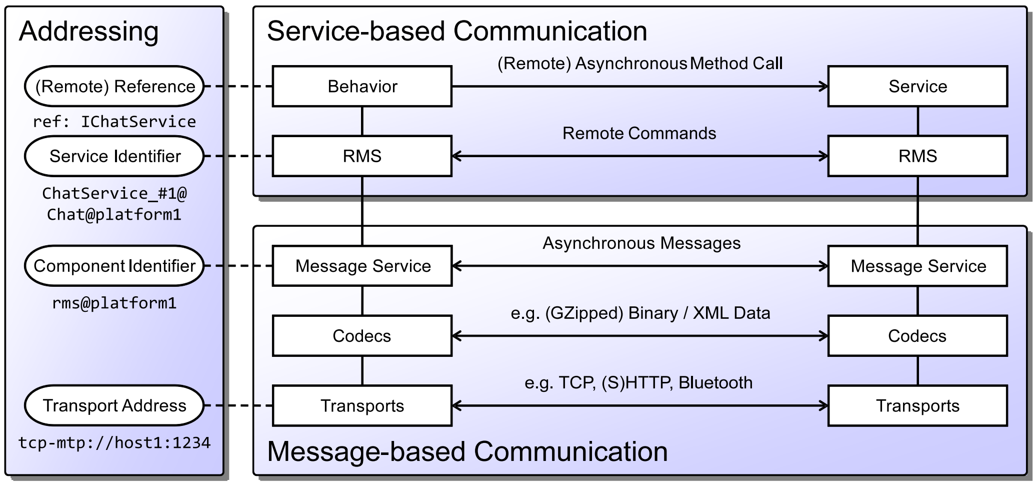


Fig. 5.2: Jadex communication stack

5.2. Distributed Infrastructure. The active components approach as well as the Jadex platform implementation aim at supporting the development of distributed applications. Therefore interactions between application parts residing on different network nodes are of particular importance due to the inherent challenges of distributed applications (cf. Section 2). The Jadex platform thus provides a number of features that facilitate using the active components approach in a distributed infrastructure and at the same time hiding many of the challenging details regarding concurrency, distribution and non-functional criteria. The general goal is that the developer should be able to focus on implementing the application functionality, based on the active components programming model. The model naturally deals with concurrency issues due to the asynchronous interaction style and the single-threaded component approach. Dealing with distribution and non-functional aspects should ideally be delayed until application deployment. In the following, first the important Jadex features with regard to distribution transparency are described. Afterwards, with security and web service interoperability two examples of supporting non-functional aspects are given.

5.2.1. Distribution Transparency. Distribution transparency is achieved by a set of different mechanisms that shield communication and discovery issues from the application developer. The communication stack is illustrated in Figure 5.2. To the left, the addressing schemes of the different layers are shown with examples. In the upper half, the high-level mechanisms for service-based communication are shown. The lower part contains the infrastructure for message-based communication. From the viewpoint of a developer, a required service is transparently bound to a local or remote reference. In case of a remote reference, the required service resolves to a proxy implementing the desired service interface, e.g. *IChatService* for a chat application. When the component behavior as programmed by the application developer invokes a method on this proxy, the call is delegated to the remote management system (RMS). Remote operations such as method invocations, callback results, as well as remote service searches are encapsulated as so called remote commands, which are exchanged between RMS components on different platforms. E.g. to perform a remote method call, a service identifier is stored in the proxy, to uniquely identify the service implementation and the corresponding remote component. The RMS at the caller side (left) uses the platform part of the service identifier to build the identifier of the remote RMS component. The remote method call command is sent as a message to the remote RMS, which uses the included service identifier to locate the component and invokes the requested method on the provided service (cf. Section 4.4). The result of the service invocation is sent back from the remote RMS using a remote result command that includes a callback identifier to match the result to the original call for updating the corresponding future (cf. Section 4.1).

The RMS requires a *message-based communication* infrastructure that allows direct exchange of asynchronous messages between arbitrary platforms. Furthermore, the messages should be able to contain arbitrary Java objects for capturing, e.g., complex method parameter values from an application domain. The management of message exchanges is implemented in the message service, which handles message contents using codecs and transmits messages with the help of transports (cf. Figure 5.2, lower half). Two types of codecs are supported. One codec type is required for (un)marshaling objects to or from a byte or character stream and the

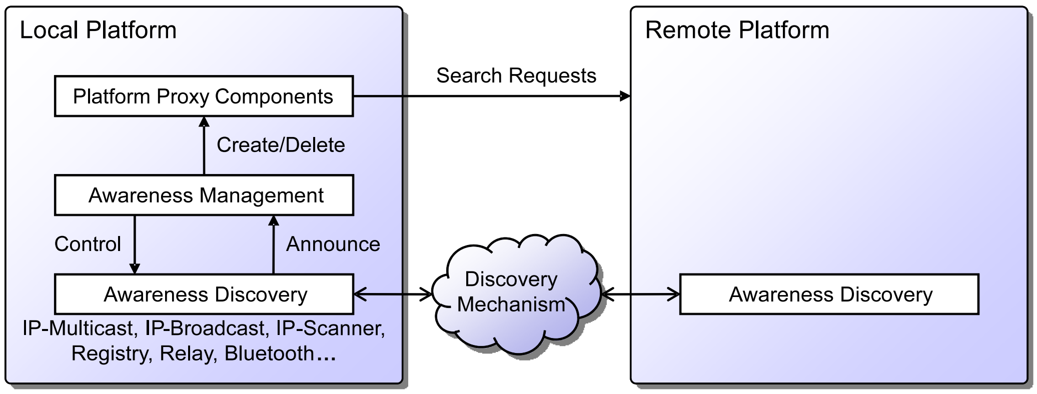


Fig. 5.3: Jadex platform awareness

other type is optional and operates on the stream for adding features such as compression or encryption. For supporting development as well as production environments, (un)marshaling can be done to a compact binary format or to a human readable XML format [14].

When sending a message, the message service collects the transport addresses stored in the component identifiers. Each transport realizes a different means for transmitting a message, e.g. using a direct TCP connection, mediation via an HTTP relay server, or forwarding in a Bluetooth scatter network. A transport also acts as receiver for incoming messages, which are passed to the message service for decoding and delivery. For each received and decoded message, the message service identifies the receiver components based on their component identifier and places the message in their inbox.

The communication stack described above achieves distribution transparency as long as some communication participants are already acquainted. E.g. when a chat component holds a remote reference to the chat service of a remote participant, communication happens transparently in response to method calls. Therefore, the programming API does not distinguish between local and remote calls (access transparency). In addition, the developer does not need to care about how the message transports reach the target platform hosting the service (location transparency). To achieve access and location transparency also for initial acquaintances, the binding of required services is transparently expanded to include remotely provided services using a so called awareness approach (cf. Figure 5.3). For this purpose, *proxy components* can be started on a local platform, that represent the remote platform. When a service is searched for on the local platform and the search scope allows including remote platforms, all proxy components on the local platform pass a search request to the RMS to issue a service search also on the corresponding remote platform. Therefore, from the viewpoint of the developer, global service searches (e.g. for binding a required service of a component) are transparently forwarded to all platforms, for which a proxy component exists locally. To discover the available platforms in the network automatically, different discovery mechanisms are available. The awareness management controls the descovery mechanisms and receives announcements of newly discovered remote platforms. It takes care of instantiating corresponding proxies for discovered platforms and also removes proxies for platforms that disappear or time out, such that only live platforms are included in service searches.

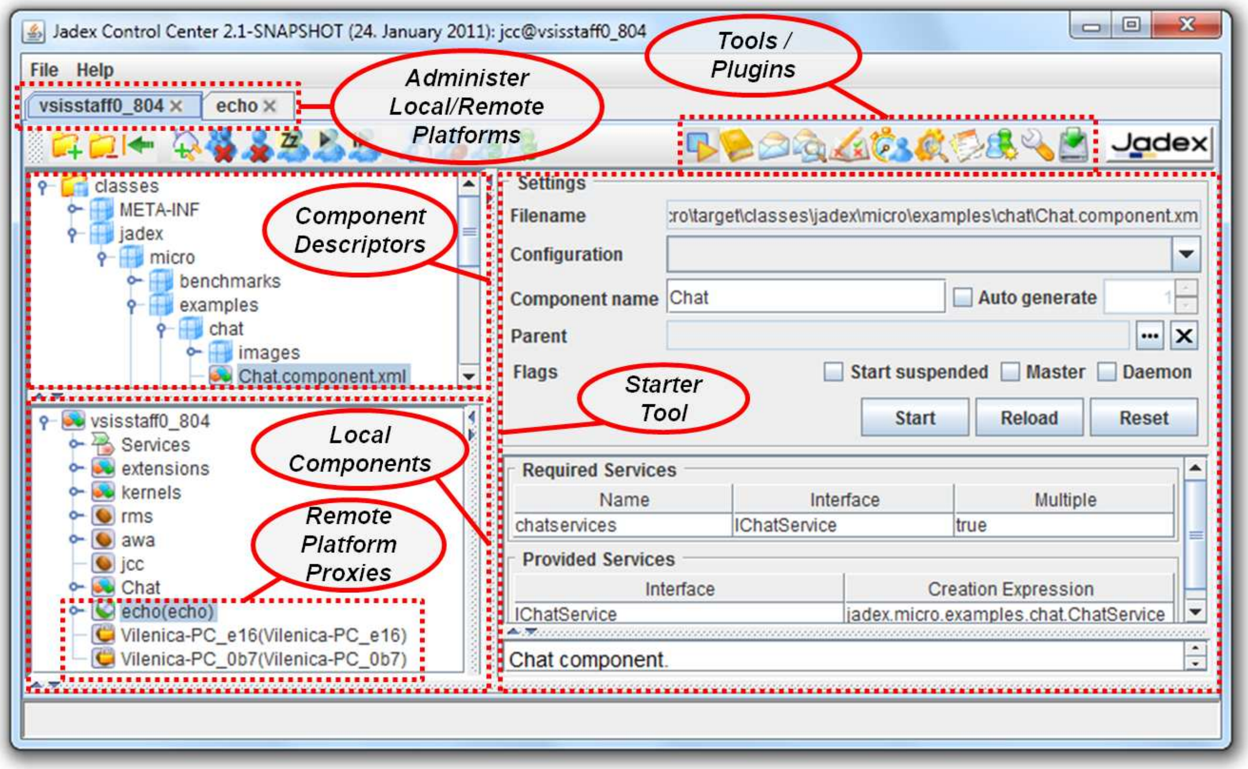
Depending on the requirements of the network, different discovery mechanisms can be employed separately or in combination. Common for all discovery mechanisms is that the same discovery mechanism needs to be running on the local as well as the remote platform. Some mechanisms are based on direct communication, such as the broadcast, multicast and scanner discovery implementations, which are well suited for local (e.g. company) networks. E.g., broadcast discovery components send and receive UDP broadcast packets containing the (remote) platform information, thus making the platforms known to each other. Unlike these direct mechanisms, other mechanisms require an intermediate, such as the relay and registry discovery approaches. They allow discovery to expand beyond local network borders and enable an internet-scale awareness. E.g. the registry discovery employs a central registry component, where all platforms announce their existance and look up other platforms. Regarding the technical implementation, the mechanisms differ whether they are based on an existing transport. E.g. the broadcast, multicast, scanner and registry are independent of any transport. The relay discovery is implemented as part of the relay transport, i.e. the relay discovery component sends a specific message through the relay transport containing the platform information. The relay server collects all platform information and sends it to other platforms, registered at the relay server. Similarly, the Bluetooth transport keeps track of the platforms participating in a Bluetooth scatter network and provides this information to the Bluetooth discovery component. Therefore, the Bluetooth transport and discovery are well suited for platforms running on mobile (e.g. android) devices connected in an ad-hoc network.

5.2.2. Non-functional Aspects. The active components approach inherits the intention from traditional component approaches to separate the implementation of component functionality as much as possible from the treatment of non-functional aspects. Ideally, non-functional aspects need be considered during implementation not at all and can be handled later during application deployment by providing appropriate component configurations. In general, the active components approach supports at least two ways of configuring non-functional aspects in a deployed application. The first way is to provide additional meta-information for specific components, either in the component descriptions or in external composite configurations. One typical use case is adapting a required service binding to the specific deployment, e.g. switching between a static wiring of components inside a composite and a dynamic open system where bindings are resolved using a global service search. The second way consists in providing different service implementations for different environments, such that both can be transparently exchanged as needed without having to touch the components that use this service. A common example would be a storage service that could be implemented as simple in-memory storage for testing, database-backed storage for medium-sized production systems and cloud storage for highly scalable applications. To support easy configuration of recurring non-functional aspects, many features of Jadex are implemented using the first or second approach, such that the developer can always adapt them to the current usage context. As an example, two features are presented in the following. The first is an extension to support web service publication and invocation and thus serves the interoperability of Jadex-based and other applications. It is realized using the meta-information approach. The second example concerns security of remote component interactions and employs annotations as well as a replaceable service.

For supporting seamless interaction between Jadex-based systems and external applications, a web service extension was realized [6]. The goal was to transparently embed external WSDL and REST web services into the active components service ecosystem and also support the publication of arbitrary active components services using a WSDL or REST interface without having to change the service implementation. The publication of services can be done using meta-information in the component description as part of the provided service declaration. Considering web service publication as a deployment issue, the corresponding meta-information can also be specified separately, e.g. when composing an application from existing components. In this case, the existing component descriptions need not be changed, as the new information is only contained in the application (deployment) descriptor. Similarly, for incorporating an external web service, a wrapper component can be added to the application, that provides the external service as a Jadex service. Therefore application components are now able to find and invoke the external service like any other service inside the application. The wrapper component maps the web service operations to an asynchronous active components service interface. In the simplest case, only this wrapper interface needs to be specified by the developer and an appropriate wrapper component is automatically generated at runtime. More complex mappings can be achieved by adding annotations to the interface or providing separate wrapper functionality (cf. [6] for more details).

Another important aspect of open distributed systems is security. When systems are technically enabled to transparently perform arbitrary remote operations, the platform administrator has to make sure that only authorized users are granted access to critical operations. In Jadex, security is handled on two levels. On the first level, general security requirements are annotated to operations defined in service interfaces. Therefore, the application programmer has to decide if a special treatment of security is necessary for a specific service or one of its operations. As a default, a very strict security setting is applied to all services not annotated otherwise, such that only local interactions are possible and any remote interactions are prohibited. On a second level, the security service inside the platform is responsible for monitoring the compliance to security settings and rejecting operations in case of security faults. The security service also processes outgoing service requests for achieving compliance to current security settings. E.g. when authentication is required, the initiating security service can sign the request before sending it, by using locally stored user credentials. The security service on the receiving side verifies the signature and accepts or rejects the request accordingly.

5.3. Tools. Besides the adequate treatment of fundamental challenges like concurrency, distribution and non-functional criteria, any practical development infrastructure also needs to take care of pragmatic aspects as well. Among the most important pragmatic aspects (besides the availability of documentation) are tool-support



Filename: :ro\target\classes\jadex\micro\examples\chat\Chat.component.xm

Configuration:

Component name: Chat ☐ Auto generate

Parent:

Flags: ☐ Start suspended ☐ Master ☐ Daemon

Required Services

Name	Interface	Multiple
chatservices	IChatService	true

Provided Services

Interface	Creation Expression
IChatService	jadex.micro.examples.chat.ChatService

Chat component.

Fig. 5.4: The Jadex control center

and integration with existing development infrastructure. The Jadex active components approach is based on existing languages, such as Java and XML. As a result, most of the productivity features of existing development environments like Eclipse, such as automatic code completion, can be used while developing active components as well. Similarly, existing build tools like Maven or continuous integration servers like Hudson/Jenkins can form integral parts of setups for developing active component applications. In addition, some extensions have been developed, e.g. an Eclipse plugin that provides consistency checking of component descriptions as well as a JUnit adapter for easy testing of active components during automated builds.

Extensive work was performed to provide adequate runtime tools that allow on the one hand the administration of deployed active component applications and on the other hand are also substantially helpful for testing and debugging during development. These runtime tools are combined into the so called Jadex control center (JCC) as shown in Figure 5.4. The JCC itself is realized as an active component, running as part of a Jadex platform and is composed of a number of tools and plugins, which are available from the toolbar at the top right. The screenshot shows the starter tool. It allows browsing component descriptions from included repositories (left) and shows see the currently running local components (bottom, left), including also proxy components, which have been started by the awareness component to represent discovered remote platforms. The starter tool further allows creating new component instances from a selected component description, by editing and starting a configuration (right). Besides the starter, a debugger tool allows inspecting the internal state of a component and executing a component stepwise. As the internal state of a component differs with respect to the component type, different debugger views are provided for, e.g. BPMN or micro components. Several other tools are mainly required for administration purposes, as they provided configuration options for basic platform functionality. E.g. the awareness tool allows to enable/disable the available discovery mechanisms and to control the creation of platform proxies with blacklists and whitelists. In another tool, the security settings can be edited, e.g. setting a local platform password or entering credentials for connecting to remote platforms.

All functionality of the JCC supports interaction with local as well as remote platforms. When the user has enough rights to administer a remote platform, she can right-click on its platform proxy, as e.g. shown in the bottom left of in the starter tool, and choose to open an additional JCC view for this platform. The currently open JCC view are shown as tabs at the top left of the JCC. In the spirit of distribution transparency, the view of a remote platform is exactly the same as that of the local platform and the user may interact with any tools, provided that the security constraints hold. Therefore the Jadex platform provides distribution transparency not only for programming, but also for testing, debugging and administration of active component applications. The practicality of the Jadex concepts, middleware and tools are illustrated in the following section using real world application examples.

6. Case Studies. The usefulness and practicality of the approach is illustrated with three case studies that have been implemented using active components. All applications have been developed together with different companies. The first application called tariff maxtrix belongs to the area of distributed calculations and is used to precompute urban traffic prices. The second application called DiMaProFi (Distributed Management of Processes and Files) is a distributed and process-driven ETL (extract-tranform-load) tool. As third application a distributed and goal-oriented workflow management system in the context of the Go4Flex project is presented. It has to be noted that due to secrecy reasons not all details of the commercial scenarios can be described.

6.1. Tariff Matrix. The company HBT[5] is responsible for a journey planner called GEOFOX that computes best routes using the local public transportation of Hamburg.[6] GEOFOX is a client server based system that allows users to use different frontends such as normal browsers as well as mobile devices such as smart phones. Besides getting information about the connection itself, GEOFOX also provides price information to the users. Tickets can then be bought via different channels including an online shop and ticket automatons. In this respect the ticket automations have to be enabled to compute the same prices as GEOFOX which is difficult due to their restricted computing power and the fact that they are not always connected to the Internet. Hence, currently an offline mechanism is used to precompute ticket prices of all possible connection alternatives. The results of this computation is expressed as a tariff matrix, i.e. a mostly undirected, fully connected graph with multi edges.[7] HBT has to recompute the matrix several times a year whenever tariff-structural or environmental changes have occurred. As matrix computation is computationally expensive HBT already uses a decentralized approach in which a divide and conquer strategy is applied to distribute work among normal company workstations.

A process analysis of existing solution revealed that the following improvement areas are especially promising. First, the amount of manual activities should be reduced and the matrix computation process should automated to a higher degree. Second, the state of processes and steps should be made more observable in order to detect problems and failures earlier. Third, downtimes in the processes should be avoided. Following these objectives a workflow driven solution based on Jadex active components has been developed and tested. The architecture of the system consists of a server agent and multiple worker agents, whereby the server coordinates work distribution and collection and the clients are responsible for computing predefined parts of the tariff matrix. Jadex supported achievement of the mentioned goals in the following way. The overall process could be modelled and implemented as BPMN workflow thus reducing many manual steps that originally existed to trigger next steps. Using active components allowed for using proactive notifications of worker agents based on service invocations instead of relying on the produced files in a shared file system. Faster information propagtion to the master gives users an up to date view of the system progress and reduces dectection times of errors. Finally, downtimes within the process can now be observed by the master and adequate reactions, such as automatically including new workers detected by Jadex awareness, can be performed.

6.2. DiMaProFI. DiMaProFi is a software product currently developed from Uniique AG[8] together with the University of Hamburg. The company is a database vendor that is specialized on data preprocessing in context of data warehousing. Most of their workflows in the area of ETL are distributed, long lasting, and interleaved with manual quality assurance tests. These properties make such workflows hard to automate and control without considerable human involvement. Existing tool support is based on centralized architectures with a designated node that controls the overall workflow. Such approach is problematic in environments with dynamically changing network setups, because e.g. spontaneous occurring network partionings or node

[5]Hamburger Berater Team GmbH, `http://www.hbt.de/`

[6]Public transport in Hamburg is managed by the company Hamburger Hochbahn AG.

[7]Between source and target multiple routes with different prices may exist.

[8]`http://www.uniique.de/`

breakdowns. Hence, the newly created DiMaProFi software solution will enable executing distributed ETL workflows modelled in a simplified version of BPMN relying on hierarchical decomposition via subworkflows and a palette of prebuilt ETL activies. Each ETL activity will be mapped to a service and can thus be executed locally as well as remotely. In the workflow description, constraints can be specified to bind the execution location to specific target nodes if this is deemed necessary, e.g. when subsequent steps of the process operate with data that should not be copied to other nodes for efficiency or privacy reasons.

Using active components as foundation for DiMaProFi simplified the system development in the following ways. One important aspect is the possibility to apply a component based design with clearly defined service interfaces. This allows to build up a set of ready to use ETL functionalities available in a network of components. In constrast to purely service oriented architecture, in which services are rather static, such services can dynamically appear and disappear by starting and stopping active components at any network node. Using the monitoring capabilities of DiMaProFi the infrastructure can react to environmental changes by dynamic reconfiguration of service providers in the network. Another important advantage of using active components consists in the automatically achieved distribution transparency. The processes and program code need not to be changed if local or remote services are used. Finally, the development of DiMaProFi also benefits from the active component property of different internal comopnent architectures. This allows using BPMN for complex processes that should be readable by customers, e.g. template workflows and basic services, and Java based micro agents for components and services with high demands regarding efficiency and compactness.

6.3. Go4Flex. The Go4Flex project is conducted together with Damiler AG and is targeted at business process management [13]. At Daimler difficulties in realizing complex business processes have been observed, especially if these processes are long running and contain a lot of different potential errors that might occur. Traditional workflow languages like BPMN are useful if workflow semantics is rather procedural and can be expressed as sequences of actions. In case of workflows with a more declarative semantics BPMN and similar languages reach their limits, as exceptional cases have to be described explicitly. For this reason, in Go4Flex a new goal-oriented modelling language called GPMN (goal-oriented process modeling notation) is developed which can be used to describe workflows in a high-level requirement driven way. GPMN uses two modelling levels. Higher-level workflows are modelled with goals, whereas lower-level workflows are modelled in standard BPMN. In this way the goal-oriented workflows form an upper control level that is used to decide which concrete BPMN workflows should be used according to the current context.

In Go4Flex active components and Jadex have been used for two purposes. First, the active component metaphor naturally allowed to execute different kinds of workflows, GPMN and BPMN, in the same infrastructure, as both kinds of workflows can be seen as active components that differ only with respect to the internal architecture used. The goal semantics of GPMN workflows has been directly mapped to the extensively studied BDI goal semantics including different goal types and inhibition relationships between goals [7, 5]. Using a model transformation approach, GPMN workflow model are converted to BDI agent representations so that at runtime the BDI agent interpreter can be resused for executing GPMN workflows. Second, as part of Go4Flex a workflow management system (WfMS) has been built relying on Jadex. In this way the workflow management system can directly profit from the characteristics of the distributed middleware by exploiting service based communications between clients and WfMS. In order to better validate the correctness of the GPMN workflows a test case driven evaluation tool has been developed. It executes a GPMN workflow for each possible combination of allowed input values and checks the results of the single runs according to predefined correctness criteria. In order to execute the possibly large number of runs efficiently the Jadex simulation support is used, leading to as-fast-as-possible execution semantics [12].

7. Related Work. As the objective of this paper is to motivate a new conceptual approach for developing distributed systems, alternative integration approaches have been categorized according to the pardigms (objects, agents, SOA, and components) they aim to combine (cf. Fig. 7.1). Additionally, the approaches have to be distinguished according to the level they address, i.e. are they rather conceptual proposals or do they combine the concepts with a middleware that follows these ideas. The figure shows that many integration approaches exist that belong to different combinations of paradigms, but none of them is targeted towards an integration of ideas from all four main paradigms. Only the work of [1] shares the same goal, but proposes a meta-model combination approach, in which the core entities of the main paradigms are brought together into a coherent scheme. In contrast our approach strives at a simplication of development by introducing a new notion that encompasses the paradigm key characteristics and also provides a middleware infrastructure that demonstrates

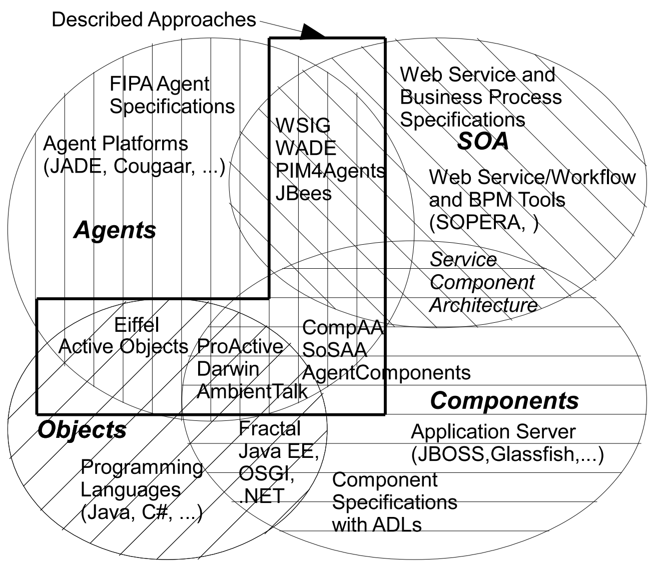

Described Approaches
FIPA Agent Specifications
Agent Platforms (JADE, Cougaar, ...)
Agents
WSIG
WADE
PIM4Agents
JBees
Web Service and Business Process Specifications
SOA
Web Service/Workflow and BPM Tools (SOPERA,)
Service Component Architecture
Eiffel
Active Objects
ProActive
Darwin
AmbientTalk
CompAA
SoSAA
AgentComponents
Components
Objects
Application Server (JBOSS,Glassfish,...)
Programming Languages (Java, C#, ...)
Fractal
Java EE,
OSGI,
.NET
Component Specifications with ADLs

Fig. 7.1: Paradigm integration approaches

its capabilities.

In the following specific combination areas and representatives from these areas will be considered in more detail. We have chosen to discuss those combination areas in which the agent paradigm is involved. In the area of agents and objects especially concurrency and distribution has been subject of research. One example is the active object pattern, which represents an object that conceptually runs on its own thread and provides an asynchronous execution of method invocations by using future return values [29]. It can thus be understood as a higher-level concept for concurrency in OO systems. In addition, also language level extensions for concurrency and distribution have been proposed. One influential proposal much ahead of its time was Eiffel [21], in which as a new concept the virtual processor is introduced for capturing execution control.

Also in the area of agents and components some combination proposals can be found. CompAA [2], SoSAA [8] and AgentComponents [18] try to extend agents with component ideas. In CompAA a component model is extended with so called adaptation points for services. These adaptation points allow to choose services at runtime according to the functional and non-functional service specifications in the model. The flexibility is achieved by adding an agent for each component that is responsible for runtime service selection. The SoSAA architecture consists of a base layer with some standard component system and a superordinated agent layer that has control over the base layer, e.g. for performing reconfigurations. In AgentComponents, agents are slightly componentified by wiring them together using slots with predefined communication partners. In addition, also typical component frameworks like Fractal have been extended in the direction of agents e.g. in the ProActive [4] project by incorporating active object ideas.

One active area, is the combination of agents with SOA [28] mostly driven by the need of dynamic service composition, i.e. agents are used to dynamically search and select services at runtime according to given requirements or service level agreements [19, 30]. These approaches mainly deal with aspects of semantic service descriptions and search but do not aim at a paradigm integration by itself. Also other SOA related integration approaches that deal with workflows and agents have been put forward. Examples are agent-based service invocations from agents using WSIG (cf. JADE[9]), or model-driven code generation approaches like PIM4Agents [32] and workflow approaches like WADE (cf. JADE) or JBees [9]. Agents are considered useful for realizing flexible and adaptive workflows especially by using dynamic composition techniques based on negotiations and planning mechanims, e.g. proposed in MASE [23].

Finally, also the combination of agent, component and object concepts have been investigated. With ProActive [3] and AmbientTalk [31] two recent approaches exist that provide sound conceptual foundations and also a ready-to-use middleware framework. ProActive is targeted towards supporting Grid environments and conceptually relies on active objects that have been extended with distribution features. The framework adds further

[9]`http://jade.tilab.com`

support for typical Grid requirements such as map-reduce support, security and reliability features. AmbientTalk has been designed to support mobile ad-hoc networks with a dynamic number of clients. It introduces a new programming language that is also based on the distinction of active and passive objects. Services of active objects are dynamically discovered and invoked with a future based invocation scheme. AmbientTalk is conceptually close to active components but does rely on a complete component model, especially provided and required services cannot be declared.

The discussion of related works shows that the complementary advantages of the different paradigms have led to a number of approaches that aim at combining ideas from different paradigms. From all areas involving agents the most prominent approaches have been evaluated. The majority of those approaches are rather technical integration attempts not targeted at devising new conceptual entities. Most relevant with respect to our works are the approaches of ProActive and AmbientTalk that both share some underlying ideas with active components. Active components extends those in the direction of agents (instead of active objects) and present a new unified conceptual model that combines the characteristics of services, components and agents.

8. Conclusions and Outlook. In this paper it has been argued that different classes of distributed systems exist that pose challenges with respect to distribution, concurrency, and non-functional properties for software development paradigms. Although, it is always possible to build distributed systems using the existing software paradigms, none of these offers a comprehensive worldview that fits for all these classes and for each class some conceptual problems usually remain unsolved. Hence, developers are forced to choose among different options with different trade-offs and cannot follow a common guiding metaphor. From a comparison of existing paradigms the active component approach has been developed as an integrated worldview from component, service and agent orientation. Based on this conceptual approach a concrete programming model has been devised, which provides concurrency support following actor based concepts. It fosters distribution transparency by not distinguishing between local and remote service as well as by hiding all aspects of service registration and search from the user. Non-functional aspects are supported on basis of meta-information that can be annotated to components as well as by adding or exchanging new infrastructure services. An example for the first category are security annotations, an example for the latter category is web service publishing. The active component approach has been realized in the Jadex platform, which includes modeling and runtime tools for developing active component applications. The usefulness of active components has been further illustrated by an application from the disaster management domain.

As one important part of future work the enhanced support of non-functional properties for active components will be tackled. In this respect it will be analyzed if SCA concepts like wire properties (transactional, persistent) can be reused for active components. Furthermore, currently a company project in the area of data integration for business intelligence is set up, which will enable an evaluation of active components in a larger real-world setting.

REFERENCES

[1] N. Aboud, E. Cariou, E. Gouardères, and P. Aniorté, *Service-oriented integration of component and agent models*, in ICSOFT 2011 - Proceedings of the 6th International Conference on Software and Data Technologies, M. E. Cuaresma, B. Shishkov, and J. Cordeiro, eds., SciTePress, 2011, pp. 327–336.

[2] P. Aniorté and J. Lacouture, *Compaa : A self-adaptable component model for open systems*, in 15th Annual IEEE International Conference and Workshop on Engineering of Computer Based Systems (ECBS 2008), IEEE Computer Society, 2008, pp. 19–25.

[3] F. Baude, D. Caromel, C. Dalmasso, M. Danelutto, V. Getov, L. Henrio, and C. Prez, *Gcm: a grid extension to fractal for autonomous distributed components*, Annals of Telecommunications, 64 (2009), pp. 5–24.

[4] F. Baude, D. Caromel, and M. Morel, *From distributed objects to hierarchical grid components*, in CoopIS, Springer, 2003, pp. 1226–1242.

[5] L. Braubach and A. Pokahr, *Representing long-term and interest bdi goals*, in Proceedings of International Workshop on Programming Multi-Agent Systems (ProMAS-7), T. Braubach, Briot, ed., IFAAMAS Foundation, 5 2009, pp. 29–43.

[6] L. Braubach and A. Pokahr, *Conceptual integration of agents with wsdl and restful web services*, in Int. Workshop on Programming Multi-Agent Systems (PROMAS'12), 2012.

[7] L. Braubach, A. Pokahr, D. Moldt, and W. Lamersdorf, *Goal Representation for BDI Agent Systems*, in Proceedings of the 2nd International Workshop on Programming Multiagent Systems (ProMAS 2004), R. Bordini, M. Dastani, J. Dix, and A. El Fallah-Seghrouchni, eds., Springer, 2005, pp. 44–65.

[8] M. Dragone, D. Lillis, R. Collier, and G. O'Hare, *Sosaa: A framework for integrating components & agents*, in Symp. on Applied Computing, ACM Press, 2009.

[9] L. Ehrler, M. Fleurke, M. Purvis, B. Tony, and R. Savarimuthu, *AgentBased Workflow Management Systems*, Inf Syst E-Bus Manage, 4 (2005), pp. 5–23.

[10] M. Fowler, *Inversion of control containers and the dependency injection pattern*, 2004. http://martinfowler.com/articles/injection.html.

[11] C. Hewitt, P. Bishop, and R. Steiger, *A universal modular actor formalism for artificial intelligence*, in IJCAI, 1973, pp. 235–245.

[12] K. Jander, L. Braubach, A. Pokahr, and W. Lamersdorf, *Validation of agile workflows using simulation*, in Third international Workshop on LAnguages, methodologies and Development tools for multi-agent systemS (LADS010), O. Boissier, A. E.-F. Seghrouchni, S. Hassas, and N. Maudet, eds., CEUR Workshop Proceedings, 8 2010, pp. 41–47.

[13] K. Jander, L. Braubach, A. Pokahr, and W. Lamersdorf, *Goal-oriented processes with gpmn*, International Journal on Artificial Intelligence Tools (IJAIT), (2011).

[14] K. Jander and W. Lamersdorf, *Compact and efficient agent messaging*, in Int. Workshop on Programming Multi-Agent Systems (PROMAS'12), 2012.

[15] P. Jezek, T. Bures, and P. Hnetynka, *Supporting real-life applications in hierarchical component systems*, in 7th ACIS Int. Conf. on Software Engineering Research, Management and Applications (SERA 2009), Springer, 2009, pp. 107–118.

[16] R. Karmani, A. Shali, and G. Agha, *Actor frameworks for the jvm platform: a comparative analysis*, in Proceedings of the 7th International Conference on Principles and Practice of Programming in Java, PPPJ '09, New York, NY, USA, 2009, ACM, pp. 11–20.

[17] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. L. J.-M., Loingtier, and J. Irwin, *Aspect-oriented programming*, in Proceedings of the European Conference on Object-Oriented Programming (ECOOP'97), 1997, pp. 220–242.

[18] R. Krutisch, P. Meier, and M. Wirsing, *The agent component approach, combining agents, and components*, in 1st German Conf. on Multiagent System Technologies (MATES), Springer, 2003, pp. 1–12.

[19] S. Liu, P. Küngas, and M. Matskin, *Agent-based web service composition with jade and jxta*, in Proceedings of the 2006 International Conference on Semantic Web & Web Services (SWWS), H. R. Arabnia, ed., CSREA Press, 2006, pp. 110–116.

[20] J. Marino and M. Rowley, *Understanding SCA (Service Component Architecture)*, Addison-Wesley Professional, 1st ed., 2009.

[21] B. Meyer, *Systematic concurrent object-oriented programming*, Commun. ACM, 36 (1993), pp. 56–80.

[22] Object Management Group (OMG), *Business Process Modeling Notation (BPMN) Specification*, version 1.1 ed., Feb. 2008.

[23] A. Poggi, M. Tomaiuolo, and P. Turci, *An agent-based service oriented architecture*, in WOA 2007, Seneca Edizioni Torino, 2007, pp. 157–165.

[24] A. Pokahr and L. Braubach, *Active Components: A Software Paradigm for Distributed Systems*, in Proceedings of the 2011 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2011), IEEE Computer Society, 2011.

[25] A. Pokahr, L. Braubach, and K. Jander, *Unifying agent and component concepts - jadex active components*, in Proceedings of the 8th German conference on Multi-Agent System TEchnologieS (MATES-2010), C. Witteveen and J. Dix, eds., Springer, 2010.

[26] A. Pokahr, L. Braubach, J. Sudeikat, W. Renz, and W. Lamersdorf, *Simulation and implementation of logistics systems based on agent technology*, in Hamburg International Conference on Logistics (HICL'08): Logistics Networks and Nodes, T. Blecker, W. Kersten, and C. Gertz, eds., Erich Schmidt Verlag, 2008, pp. 291–308.

[27] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture*, vol. 2, Patterns for Concurrent and Networked Objects, John Wiley and Sons, 2000.

[28] M. Singh and M. Huhns, *Service-Oriented Computing. Semantics, Processes, Agents*, Wiley, 2005.

[29] H. Sutter and J. Larus, *Software and the concurrency revolution*, ACM Queue, 3 (2005), pp. 54–62.

[30] M. Vallee, F. Ramparany, and L. Vercouter, *A Multi-Agent System for Dynamic Service Composition in Ambient Intelligence Environments*, in The 3rd International Conference on Pervasive Computing (PERVASIVE 2005), 2005.

[31] T. Van Cutsem, S. Mostinckx, E. G. Boix, J. Dedecker, and W. De Meuter, *Ambienttalk: Object-oriented event-driven programming in mobile ad hoc networks*, Chilean Computer Science Society, International Conference of the, 0 (2007), pp. 3–12.

[32] I. Zinnikus, C. Hahn, and K. Fischer, *A model-driven, agent-based approach for the integration of services into a collaborative business process*, in Proc. of AAMAS, IFAAMAS, 2008, pp. 241–248.

Edited by: Marcin Paprzycki
Received: May 1, 2012
Accepted: June 15, 2012

Scalable Computing: Practice and Experience
Volume 13, Number 2, pp. 121–137. http://www.scpe.org

ISSN 1895-1767
© 2012 SCPE

MSMAS: MODELLING SELF-MANAGING MULTI AGENT SYSTEMS

EMAD ELDEEN ELAKEHAL[*] AND JULIAN PADGET[†]

Abstract. Although Multi Agent Systems (MAS) have attracted a great deal of attention in the field of software engineering, with their promise of capturing complex systems, they remain far away from commercial popularity mainly due to the accessibility of MAS methodologies for commercial developers. In this paper we present a practical method for developing self-managing MAS that we believe enables not only software developers but also business people beyond the academic community to design and develop MAS using familiar concepts. We present the main three phases of the proposed methodology, with details and examples of all the visual models, followed by details of its supporting metamodel, in which we describes the MAS concepts used and their relationships. In particular, the methodology features 1. a formal specification mechanism for system norms 2. offers organizational support of MAS through institutions, and 3. supports self-management explicitly through dynamic planning.

Key words: Multiagent systems, institutions, MAS development methodology, self-management

AMS subject classifications. 68T42 agent technology, 68Uxx computing methodologies and applications

1. Introduction and Problem Statement. The main aim in multiagent systems is to provide principles for the building of complex distributed systems that involve multiple agents and to take advantage of the mechanisms for cooperation and coordination in these agents' behaviours. However, building multiagent applications for complex and distributed systems is not an easy task [9]. Add to that the development of industrial-strength applications, which requires the availability of appropriate software engineering methodologies. Although there are several good MAS development methodologies such as those reviewed in section 2, we contend that none of them stands out as both a comprehensive and a business-accessible methodology. MAS exhibit all the traditional problems of distributed and concurrent systems, and in addition bring difficulties arising from flexibility in requirements and sophisticated interactions [23], all of which result in making it very challenging to define a MAS development methodology. It was stated in 2008 in the Agentlink Roadmap that "One of the most fundamental obstacles to the take-up of agent technology is the lack of mature software development methodologies for agent-based systems." [17], and we believe this remains largely true still today.

It is unfortunately apparent that none of the existing MAS development methodologies has seen main stream take-up. We put forward some reasons of our own for this lack of impact, as well as observations from some analyses in the literature, to identify those we regard as the most crucial:

1. Lack of support for inexperienced developers; typically, a methodology will require a good knowledge of agent concepts, because developers need to specify all the semantic components of their agents. This presents a significant barrier for commercial applications and may explain why adoption of the MAS paradigm is rarely encountered.
2. The absence of an holistic view of the system logic and its cognitive aspects; this has the potential to lead to confusion and ambiguity in both analysis and design phases.
3. Incomplete coverage (within a single methodology) of the development life-cycle phases; some offer design and analysis tools but no support for deployment, while others offer theory but supporting tools may not (yet) exist.
4. The gap between design models and existing implementation languages [19]; this leads to great difficulty for programmers, as they try to map the complex designs into executable code.
5. Support for implementation; many current methodologies do not include an implementation phase and of the ones that do, such as Tropos [3], its implementation language does not explain how to implement beliefs, goals and plans, nor how to reason about agent communication.
6. Lack of complete formal representation of MAS concepts; we note the of work by Wooldridge [25], and Luck [16], but observe that neither of these can be considered complete. Even though a partial approach may be effective, the question remains, which concepts to formalize? And what is the best way to specify and describe them?

Our proposed methodology, Modelling Self-managing Multi Agent Systems (MSMAS) is intended to solve most, if not all, of the above issues. It also aims at being more accessible to a wider range of academics and

[*]Department of Computer Science, University of Bath, UK (emad@bookdepository.co.uk).
[†]Department of Computer Science, University of Bath, UK (jap@cs.bath.ac.uk).

software engineers through the support of a formal representation and semi-formal visual modelling, specified through the metamodel.

The purpose in defining a metamodel is to describe the concepts of MSMAS and to highlight their relationships, as a first step in making any system modelled using MSMAS easily transferable to any MAS technology.

The metamodel represents organizational concepts explicitly and uses dynamic planning in order to support self-management. In the following sections we highlight related work in section 2 then we present an overview of the proposed methodology and give some details of its three phases with the inclusion of a sample diagram of each model in section 3, followed by the details of the metamodel in section 4 and finally we conclude the paper and discuss future work in section 5.

2. Related Work.

In this section we begin with a short review of some of the most prominent MAS methodologies: GAIA [24], MaSE – Multiagent Systems Engineering [6], Prometheus [18], and Tropos [3], amongst others. Then we outline and evaluate two particular metamodels that we consider offer the most complete efforts at defining a unified metamodel for MAS, namely the work of Hahn et al [13] and Beydoun et al [2].

2.1. Selected MAS Development Methodologies.

GAIA Methodology: GAIA [24] is a general methodology that supports both micro (agent structure) and macro (organisational structure) development of agent systems. It was proposed by Wooldridge et al in 2000 and subsequently extended by Zambonelli et al. [26] to support open multi-agent systems. It has two phases that covering analysis and design. GAIA is both lengthy and complex in these respects, as well as lacking an implementation phase.

MaSE Methodology: Multiagent Systems Engineering (MaSE) [6] covers the full development life cycle from an initial system specification to system implementation. It has two phases, comprising seven steps in all and offers tool support in all phases. MaSE does not enforce the use of any particular implementation platform, but it has a steep learning curve for inexperienced users.

Prometheus Methodology: Prometheus [18] aims to be suitable for non-expert users to design and develop MAS. The methodology has three phases: 1. System Specification, 2. Architectural Design, and 3. Detailed Design. Prometheus has a tool that supports the design process, including both consistency checking and documentation generation. Although Prometheus is more practical than many other approaches it does not fully connect the system model to an execution platform.

TROPOS: Tropos [3] distinguishes itself from other methodologies by paying great attention to the requirements analysis, where all stake-holders requirements and intentions are identified then analysed. The modelling process consists of five phases and uses JACK for the implementation, the developers would need to map the concepts in their design into JACK's five constructs. Tropos offer some guidelines to help in this process, but it seems very lengthy and complex.

2.2. PIM4Agents: A Platform Independent Metamodel for Multiagent Systems.

The principles of the Model Driven Development (MDD) Framework of the Object Management Group (OMG)[1] define how a visual, model-based approach could be used to integrate a number of technologies used in software development. Aiming to follow this approach, Hahn et al [13] examine various multiagent metamodels including Aalaadin, ADELFE, Gaia and PASSI, then propose a unified MAS metamodel by merging the metamodels of ADELFE, Gaia and PASSI to cover all of their aspects. The problem with this approach is that it does increase the complexity of modelling MAS systems and although a unified metamodel can help in moving MAS towards a standard form, it makes it difficult to satisfy some circumstances where a specifically focused structural view of a MAS is needed.

Hahn et als' unified metamodel for MAS adopts multiple points of view to cover all the features in the different technologies. These views we now summarise:

1. **Multiagent View:** focusses on the main components of any MAS; this covers agents, their capabilities and the primary concepts of any MAS such as cooperation, interaction and environment.
2. **Agent View:** describes each individual agent and the capabilities it uses to achieve its tasks, as well as the roles it plays in the context of the MAS.
3. **Behavioural View:** focusses on plan composition, specification of how atomic tasks are done and how data flows within system control constructs.

[1]Object Management Group: `http://www.omg.org/`

4. **Organisation View:** defines the organisational structure and how cooperation is achieved between the system's individual autonomous entities.
5. **Role View:** identifies the functional states of the system's autonomous entities and their social relationships.
6. **Interaction View:** describes how autonomous entities interact and the form of this interaction.
7. **Environment View:** describes the different kinds of resources that are created by the agents or shared by the organisations.

Hahn et al propose a complete model, called PIM4Agents, that defines the abstract syntax of a domain-specific modelling language for MAS (DSML4MAS), as well as a definition of the model transformations from Platform-Independent Model (PIM) to Platform-Specific Models (PSM), which allows for the transformation of the designed model into an implementation for a specific platform such as JADE or JACK.

2.3. FAML: A Generic Metamodel for MAS Development. Motivated by the advances in Model-Driven Development [1] and with the mission of combining all different metamodels in the domain of MAS, Beydoun et al [2] attempt to develop a unified metamodel to allow for interoperability, better understanding and better communications between researchers. FAML was created following a four-step process: (i) Determination of the full set of general concepts relevant to any MAS and its model, (ii) Short-listing the candidate definitions, (iii) Reconciliation of definition differences to build a consistent set of metamodel terms, and (iv) Designation of the chosen concepts into two sets: design-time and runtime, where the central design-time concept is the system as an agent-oriented system while the central runtime concept is the environment wherein agents reside. In FAML, an agent has internal and external concepts, and the classes that relate to the agent's internals at design-time are called *agent definition level*, while those that relate to the agents internal aspects at runtime are called *agent level*. Classes that relate to the agents external aspects at design-time are called *system level*, while classes that relate to agent external aspects at runtime are called *environment level*. In summary:

1. **Design-time aspects**: Views the system as an agent-oriented structure that satisfies both functional and non-functional requirements. Roles are also used to describe the system. They are normally related to tasks either as responsible for a task or as a collaborator in a task.The InitialState concept is used to initialize the concept of AgentDefinition that is used only at runtime. An AgentDefinition consists of an initial state and a number of plan specifications, and each is composed of a number of action specifications. A system also has facet definitions: these are the aspects of the environment with which the agents can interact.
2. **Runtime aspects**: The environment is an essential part of the system: it is where the agents reside and it provides the facets they need to interact. In FAML, the environment has a history which is a composition of all message events and facet events that occurred in the environment. Agent internals at runtime comprise the collection of beliefs, desires, and intentions an agent can hold including support for basic BDI concepts, but those are not compulsory. Finally the actions that make an agent plan can be facet actions or message actions.

Both FAML and PIM4Agents offer a generic approach to the description of any MAS, and have a comprehensive metamodel that fits all views and approaches of developing MAS. As such, they are a great contribution that have together helped us to verify that our concepts could be mapped to other metamodels. However, we find that the inclusion of so many concepts in FAML has led to an increase in complexity and introduced a steep learning curve to be faced by new user. To address this issue and make the modelling process more straightforward, we have intentionally designed MSMAS to offer a carefully selected subset of MAS concepts that are essential to the description of any MAS, but especially those concepts that are more aligned to, and commonly used within, a business context.

3. MSMAS Methodology Overview. MSMAS consists of three phases that cover the full life cycle of multiagent systems development. The elements of each phase and the connections between them are show in Figure 3.1. We now outline each of the phases:

1. The first phase focusses on **System Requirements** gathering: this allows the system designer to describe many possible use case scenarios as well as to specify a high-level system goals. It has two diagram types; the **System Goals Diagram** and **Use Cases Diagrams**.
2. The second phase focusses on **Detailed Analysis and Design**; during this phase the system requirements can be transformed into a complete system model. Each diagram contributes to the building of the system **Metamodel**, which is the basis for generating all the MAS code for one or more tar-

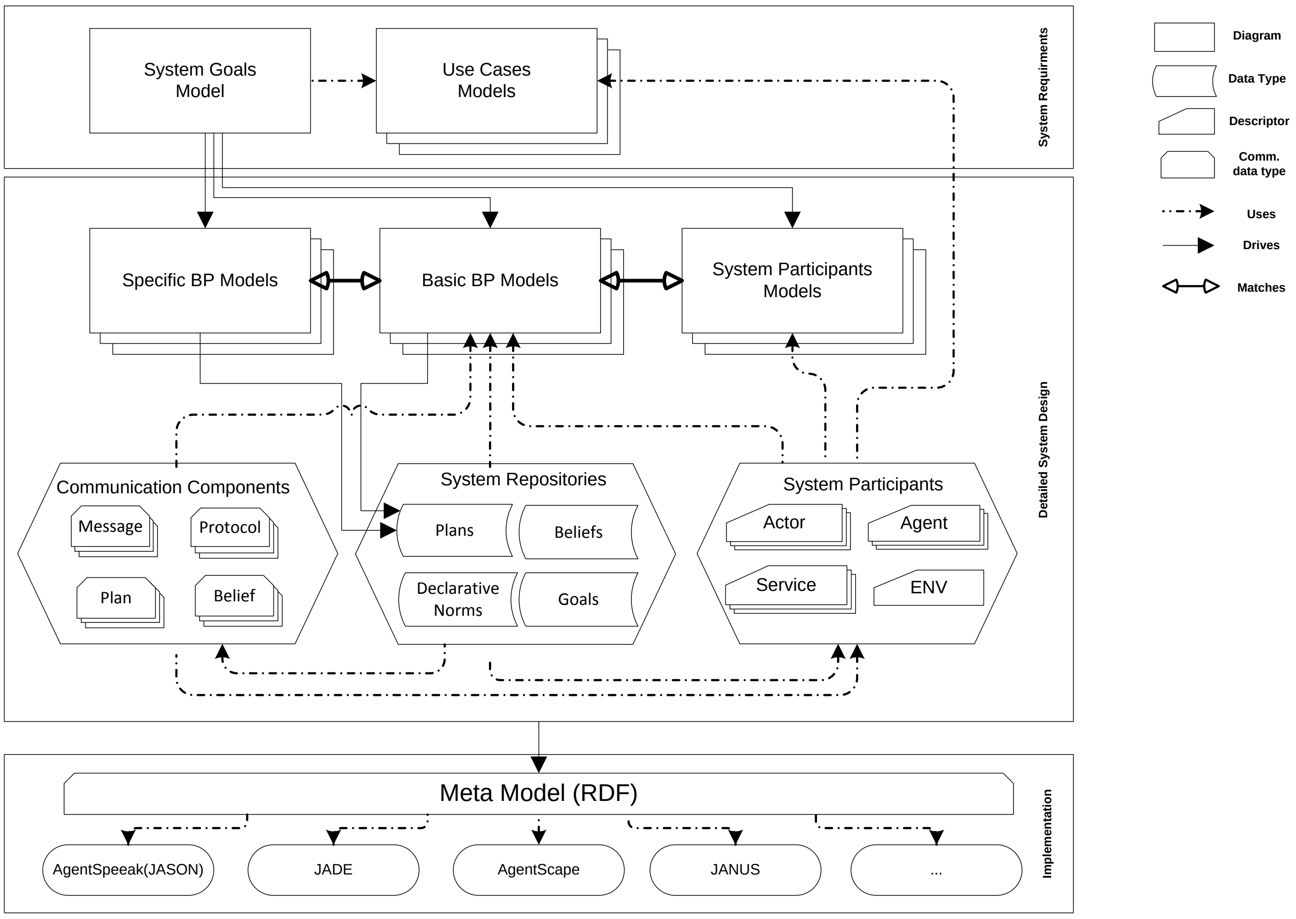


Fig. 3.1: MSMAS Overview

get execution languages/platforms. The system designer can start this phase either from the business process (BP) view or the system participant view. The BP view requires the completion of **Specific Business Process Models** and **Basic Business Process Models** diagrams. Experienced users with a good knowledge of the multiagent paradigm can alternatively start from the **System Participants Models** and the definition of their entities **(Agent–Actor–Service–Environment)** as well as the definition of communication components **(Protocol–Message)** alongside the usage or definition of **(Goals–Plans–Norms–Beliefs)**.

3. Finally, the third phase is the implementation and execution phase where the user can verify the system design and export the **Metamodel** file as RDF or choose to generate code in one of the supported execution languages/middlewares such as Jason, AgentScape etc.

In the following sections we describe each of these phases and their modelling diagrams/components in more detail and give some examples. The examples are based on elements of the inventory system described in [10]. This system comprises service provider agents (SPA), administrator agents (AA) and a central database agent (CDBA) whose task jointly is to meet the following requirements: (i) The SPA must be able to publish/update its stock-level information in the central database (ii) The SPA must provide stock-level information about any number of its catalogue items on request from another agent (iii) The SPA must report any errors in transaction with other agents (iv) The SPA must report any suspected data transmission failure (v) The AA must log and may take corrective action in response to an error reported by a SPA. The processes discussed later mostly concern the handling of the Supplier Stock Level File (SSLF), from getting a new version from a supplier to its publication in the central database.

3.1. System Requirements Phase. The purpose of this phase is to describe the system functions in terms of use cases after identifying the main system goals. There are only two models to be created during this

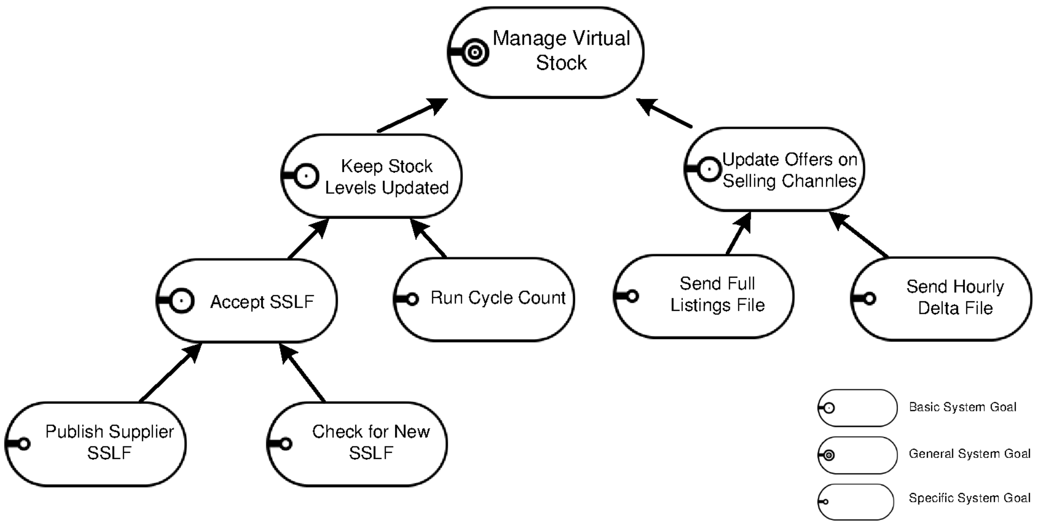


Fig. 3.2: System Goals Diagram

phase: **System Goals Model** and **Use Cases Models**.

3.1.1. System Goals Model. Every system should have a set of goals; these are simply the motivations for building the system: the system designer does not need at this stage to specify the goals in great detail, rather the goal hierarchy should be constructed until it reaches the level of detail whereby every goal can be fulfilled by *only* one basic business process. Systems goals are the drivers for all the diagrams of the next phase.

The system goal is basically the system status it is wished to achieve. The system goals definition should not to be confused with the common agent goals: in our model the system goals are procedural, in other words the goal name is similar to a method in a traditional programming language. This is very useful to divide—if we take a top to bottom approach—the system from one unit to a group of functions. At the same time, it helps to show how a particular group of actions may lead to the fulfilment of a single big system function. Figure 3.2 shows an example System Goals Model.

The system goals model contains three types of goals (i) **General System Goal:** Any system should have *only* one General System Goal. This should express the major reason for building the system. (ii) **Specific System Goals:** These are more functional goals that can be achieved by one or more business processes; each Specific System Goal can have a number of sub-goals. (iii) **Basic System Goals:** These are leaves of the goals tree; they cannot have sub-goals.

3.1.2. Use Cases Models. Use cases function as a clarification of some or all the system functionalities, in this step the system designer can create some models of the most important functions for future reference. The use cases are used in our methodology to help the system designer to think through the different functions of the system and possible issues to be considered. The use case diagram normally shows how different system participants interact, or which steps they take to carry out a system function.

Figure 3.3 shows an example use case diagram, where there are two system participants: a software agent (Supplier) and software service (Translator), and four functions. The arrows show the sequence of execution and the connectors between the agent and the function define responsibilities.

3.2. Detailed System Design Phase. The aim of the Detailed System Design phase is to define all the system components, their detailed structure and the ways they can interact with each other. There are three different diagram types in this phase; **Specific Business Process Models**, **Basic Business Process Models**, and **System Participants Models**. To complete these diagrams, the system designer needs to define/use different types of supporting entities, which are held in the form of repositories or standard descriptors.

The system designer starts this phase either by: (i) modelling the system participants; this requires some experience and familiarity with MAS concepts, or by (ii) modelling the Business Processes; this is the more common approach for business users who may not be able to define system agents and their plans etc.

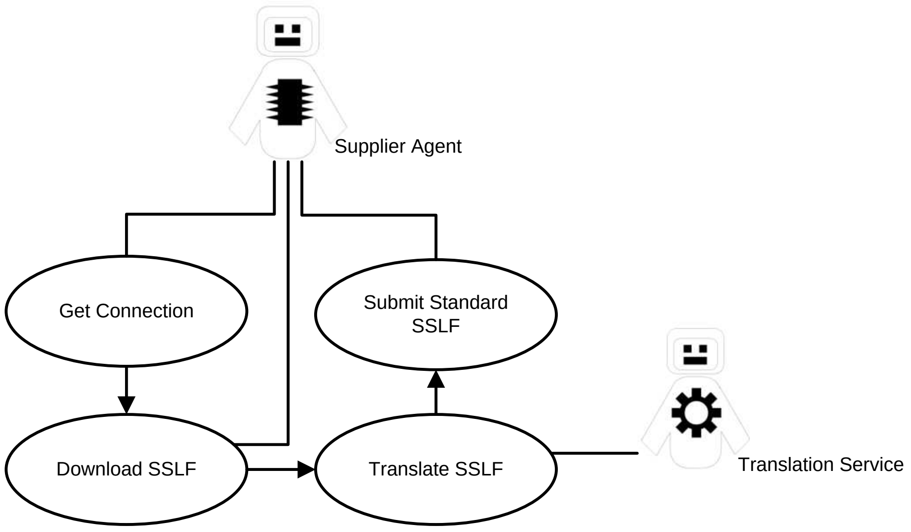


Fig. 3.3: Use Case Diagram (Publish Supplier Stock Levels File (SSLF)

3.2.1. Business Process Models (BPM). Generally, business process modelling is a way of representing organizational processes so that these processes can be understood, analysed, improved and enacted. The drawback to most BPM techniques is their procedural nature, which can lead to over-specification of the process, and the need to introduce decision points into execution that are hard to know in advance and unsuited to MAS modelling. We use declarative style modelling to describe our BPs using the Declarative Service Flow Language (DecSerFlow) [20]. More details of this are given in section 3.2.2.

Business Process Models (BPMs) are derived directly from the system goals and they are used to describe and identify the steps needed to achieve one or more of the system goals, these steps form the system plans. For each **Specific System Goal** there is at least one BPM. Each Sub-Specific Goal is represented as an **Activity** inside its Super Goal BPM. Business Process Models are either **Specific Business Process**—that is, derived from a Specific System Goal—or **Basic Business Process**, that describes a Basic System Goal.

3.2.2. Modelling BPs and Specifying System Norms Using DecSerFlow. According to Jennings [14] Commitments and Conventions Hypothesis: all coordination mechanisms can ultimately be reduced to (join) commitments and their associated (social) conventions. Introducing conventions to the system participants' interactions can be achieved through one of three approaches (i) reducing the set of possible options by restricting and hard coding all these conventions in all agents, (ii) enforcing these conventions at the protocol level that all system participants follow, so there is no way for the agent to violate the conventions even if it tries to – known as regimentation, as in [11] – or (iii) using the norms only to *influence* the system participants' behaviour as suggested by Dignum et al [7] – known as regulation.

We adopt a declarative style for modelling our BPs, namely DecSerFlow as proposed by van der Aalst and Pesic [20], which offers an effective way to describe loosely-coupled processes. Consequently, instead of describing the process as a directed graph, where the process is a sequence of activities and the focus of the design is on "*how*", the designer is able to concentrate on "*what*", by adding constraints in the activities model, as well as rules to be observed at execution time. For constraint specification, DecSerFlow uses LTL (Linear Temporal Logic) as the underlying formal language and these constraints are given as templates, that is as relationships between two (or more) activities. Each constraint template is expressed as a LTL formula. We use DecSerFlow notation and its underlying LTL formal representation.

The system designer can add convention norms in one of the following ways: (i) at the business process level, where the designer may include any number of activities alongside the business process activities and enforce any relation s/he might see necessary among the activities, or (ii) at the activity level, where the designer may choose to add the convention norms as preconditions for any number of activities; in this way the system participant would not be able to execute such activities in the absence of the satisfaction of that precondition.

3.2.3. Specific Business Process. Each system goal is realised through one specific BP, which is a collection of sub-processes or activities that normally lead to the achievement of that specific goal.

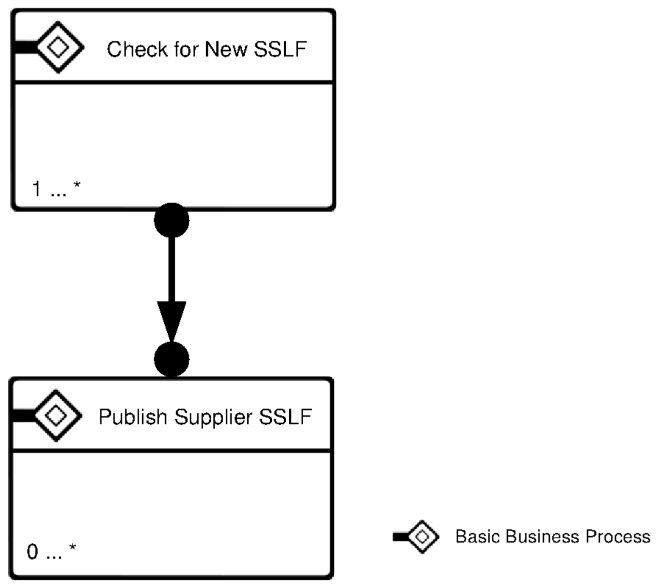


Fig. 3.4: Specific Business Process Diagram (Accept SSLF)

Figure 3.4 shows an example diagram of the "Accept SSLF" – where SSLF is the Supplier Stock Levels File – Specific Business Process, which has two activities: (A) "Check for New SSLF" that is a sub-process to achieve the "Check for New SSLF" Specific System Goal and (B) "Publish Supplier SSLF" Basic Business Process to achieve "Publish Supplier SSLF" Basic System Goal. Both activities can run an arbitrary number of times, however the **Succession relationship** requires that every execution of activity A should be followed by the execution of activity B and each activity B should be preceded by activity A. That relationship is formally expressed in LTL as: $\Box(A \Rightarrow \Diamond(B)) \wedge \Diamond(B) \Rightarrow ((\neg B) \sqcup A)$

3.2.4. Basic Business Process. A Basic Business Processes is the most detailed BP model; it can contain any number of plans for the purpose of achieving just one Basic System Goal. The Basic Business Process diagram comprises a set of activities. Figure 3.5 shows a diagram for the "Publish SSLF" Basic Business Process, which has five possible activities; each activity is carried out by one or more system participants. Each activity has its own pre-conditions and post-conditions. There is no need to specify the execution sequence, because the activity whose pre-conditions are met should start automatically. "Get connection" has no pre-conditions, which means it should start as soon as this "Publish SSLF" Business Process starts. There are two activities named "Download SSLF", each of which has the same post-conditions but a different pre-condition. During execution, based on the available resources, the supplier agent can download the new SSLF from either a FTP or an Email account. To avoid duplication of execution of this activity, there is the **not co-exists** relationship that means only one of the two tasks "A" or "B" can be executed, but not both. The not co-exists relation is expressed in LTL as: $\Diamond(A) \iff \neg(\Diamond(B))$

3.2.5. System Participants Models. System Participants Models are equivalent in context to Detailed Business Process Models. They offer a different view of the process by describing the detailed activities from the participants' perspective. They define also how activity owners communicate with other participants. System Participants Diagram includes one box for each system participant (**Agent–Actor–Service**) and one for the **Environment**. This last allows for the definition of any external event caused by other system participants.

Figure 3.6 shows an example system participants diagram for the Publish SSLF Basic BP. There are two Software Agents (Supplier Agent and Central Virtual Stock Agent) and one Software Service (Translation Service) and the Environment. The **Software Agent** is a piece of automated software that has its own set of goals expressed as states that it tries to achieve continuously. It holds its knowledge as a belief set and it is able to define new goals dynamically and update its belief set as well as define the needed steps (plans) to achieve its goals. The software agent is situated within an **Environment** that allows the agent to carry out its dynamic actions (plans), the environment also facilitates the ways in which the agent might need to communicate with other software entities sharing the same environment. We also adopt the concept of a human

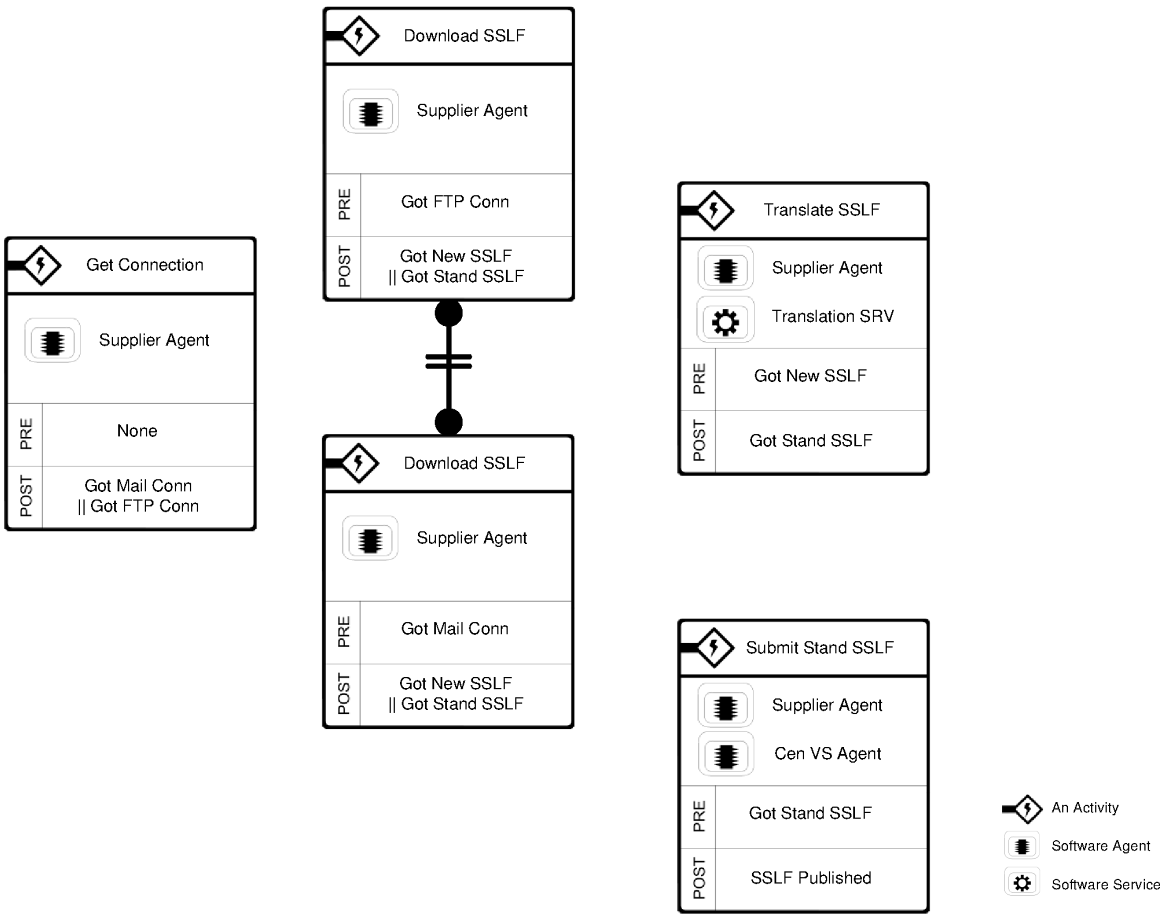


Fig. 3.5: Basic Business Process Diagram (Publish SSLF Business Process)

system participant (**Actor**), as proposed by [22] to allow for modelling a participatory team of software agents and human actors. This view is found to be more practical to support real-life scenarios where some decisions are necessarily assigned to humans to make. Finally, the system participant can be a **Software Service**, which is a piece of software that has a set of related functionalities together with policies to control its usage and is able to respond to any relevant requests from other software entities in a reactive manner.

System participants communicate using a **Communication Protocol**, which is a set of rules determining the format and transmission of a sequence of data in the form of messages between two or more system participants. MSMAS offers a number of pre-defined native protocols, as well as allowing the user to define custom protocols.

During the detailed system design phase the user can define each entity from scratch or connect it with a definition file. All entity definitions are stored in the system repositories that hold ***System and Agent Plans***, ***Environment and Agents beliefs***, ***System and Agents goals***, as well as all ***system processes***, ***communications protocols***,*and* ***system declarative norms***.

3.3. Implementation Phase. The third and final phase of MSMAS addresses verification and consistency checking across all system models. The verified system model can be exported into one **Metamodel (RDF)** . That metamodel is used to generate code for one of the supported execution languages, platforms or middlewares.

4. MSMAS Metamodel. MSMAS aims to allow the system designers to provide the absolute minimum description of a MAS that satisfies the answers to the following questions:

1. What is the purpose of building the system and its individual components? This question is answered by the definition of **Systems Goals** as the first core concept.
2. How can the system achieve its goals? The answer lies within the second core concept which is the

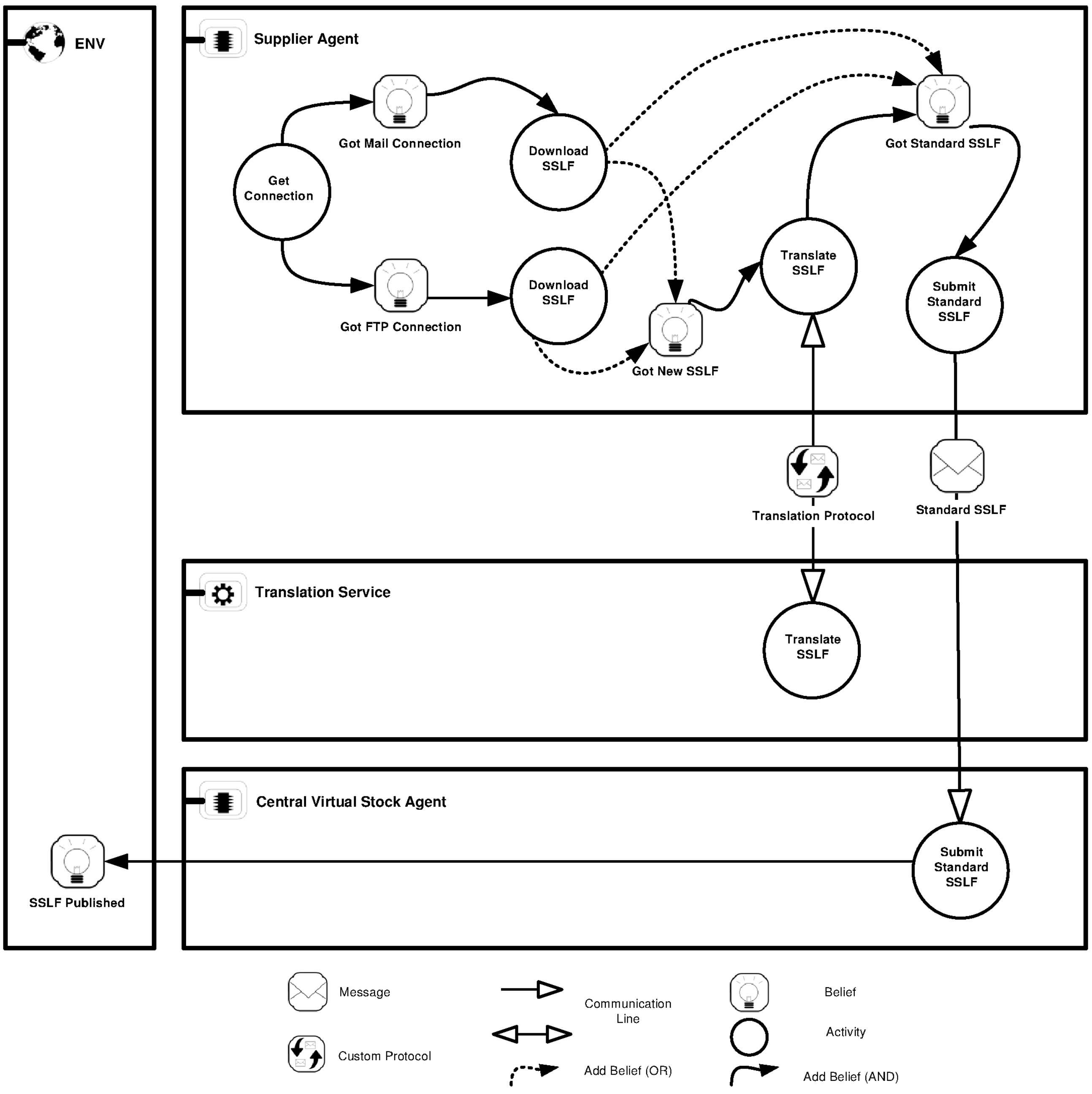


Fig. 3.6: System Participants Diagram (Publish SSLF)

system **Business Processes** and their activities.
3. Who or what is responsible for the execution of each business process activity or each complete business process of the system? The question is answered through the definition of the third core concept of **System Participants**.

Besides these core concepts, one can argue that an important objective of a MAS is the ability of more than one individual agent to interact and cooperate in favour of achieving common goal(s), or avoiding a conflict that could turn into an inability to achieve their own individual goals. To design a MAS that is able to demonstrate such behaviour, it must have an organization mechanism so its members can be regulated and so they can follow specific interaction protocols, or sets of norms. Implementing the organization mechanism can be done either by integrating it within the agents themselves or designing the organization externally to the agents, or a combination of the two [4]. Designing the organization mechanism external to the agents is normally motivated

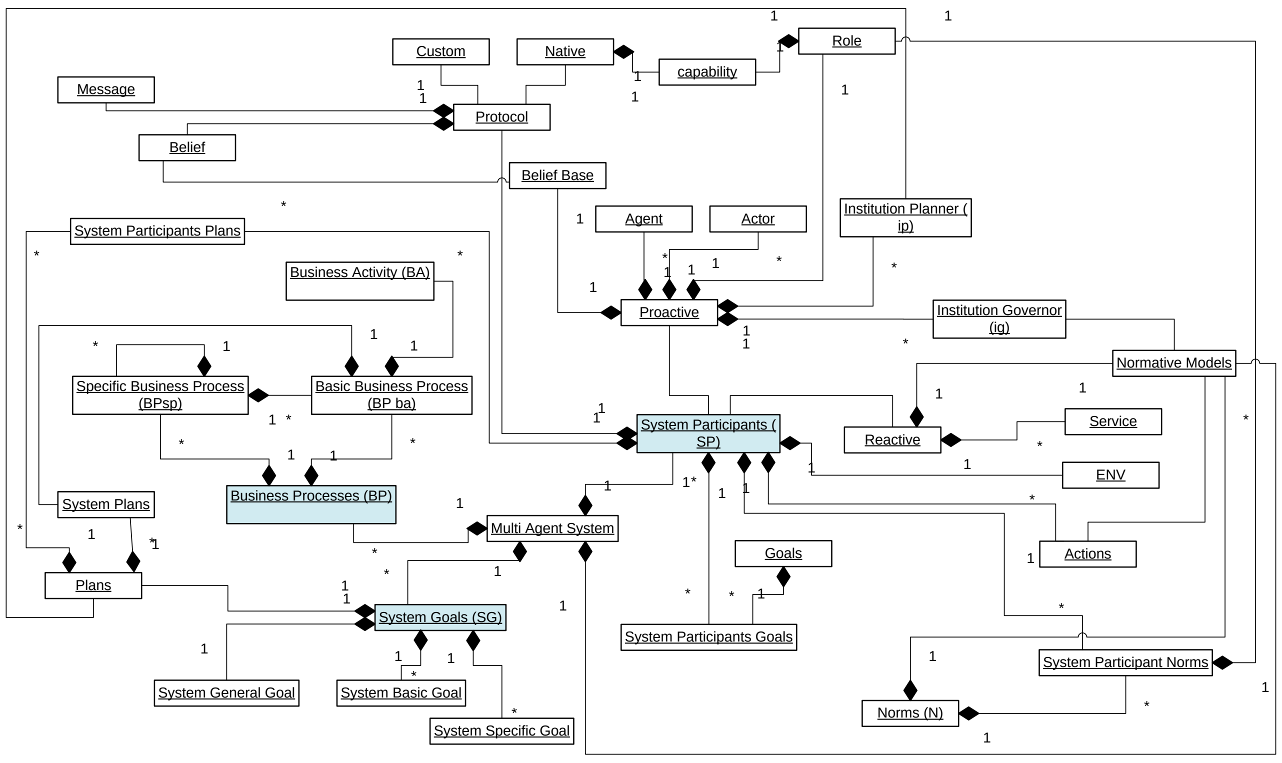


Fig. 4.1: MSMAS Metamodel

by the desire for an explicit organizational model, normative system or institution, where agents are designed to be regulated by a set or norms, which are used by the individual agents to decide how to behave or are enforced through monitoring and punishment mechanisms. In our approach we have chosen to implement the organization mechanism explicitly to allow for easier management and transformation of our metamodel into any target programming language, whether that language provides programming constructs to implement the social concepts or not. Another advantage of our approach is that it allows any agent to play any role, as long as it does maintain the capabilities needed for such role and its individual goals are achieved through playing such role [21].

In MSMAS we have chosen to support the concept of institution explicitly, so the system designer should specify the different components of an institution during the design process. This concept is described in detail in section 4.4.

Finally a true MAS needs to be an adaptive system, that is a system that can dynamically respond to the changes in its observed environment states and can evolve and find new plans that utilise its resources, as well as lead effectively to the satisfaction of its goals. The **Dynamic Planning** feature is the final main concept and discussed in more detail in section 4.5.

Figure 4.1 shows the full MSMAS Metamodel, where these three core concepts are highlighted, and Figure 4.2 is a focused view of the core concepts with their immediate sub-concepts. In the following subsections we define these core conceptual areas and their related concepts.

4.1. System Goals. The system goal is the type of the goal class, which is a state of the world that the system or any of its participants wishes to achieve. The system goals in our model are procedural, in other words the goal name is similar to a method in a traditional programming language. This is done deliberately to allow the division of — taking a top to bottom approach — the system from one unit into a group of functions. At the same time, this helps to see in a simple way how a particular group of sub-goals may lead to fulfilling one greater system goal. Each system goal may be achieved by one or more plan. Table 4.1 collects the definitions of all concepts supporting System Goal concept. As shown in Figure 4.3, the system goal is one of three types (i) **General System Goal** (ii) **Specific System Goal** (iii) **Basic System Goal**

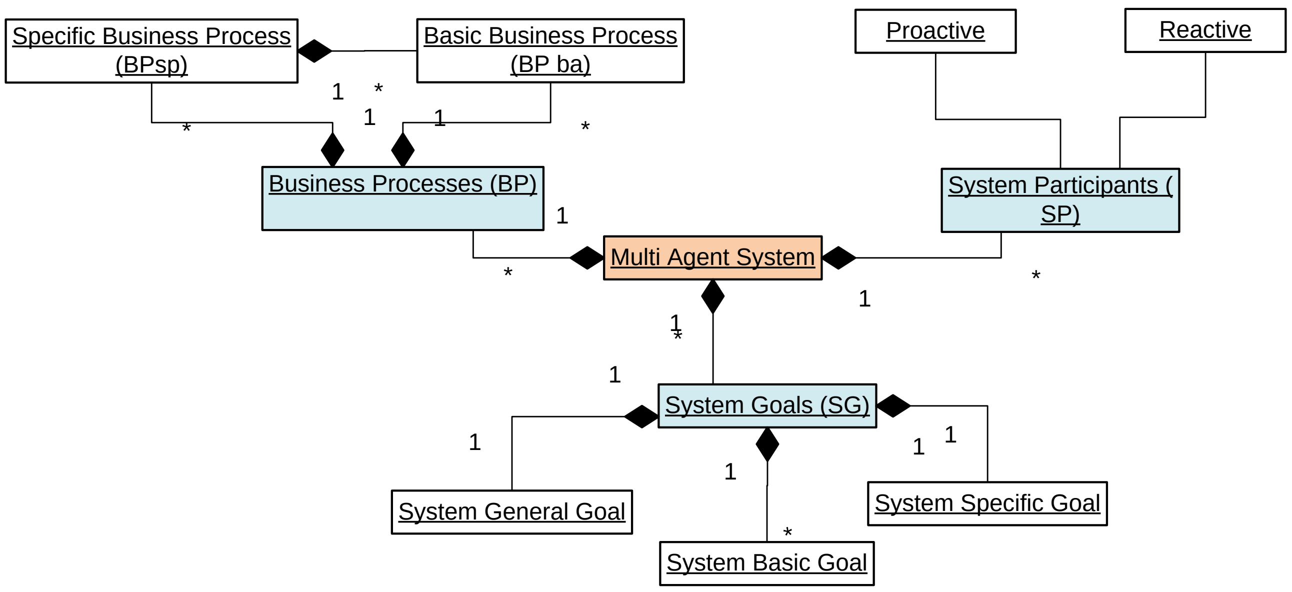


Fig. 4.2: Core MAS System

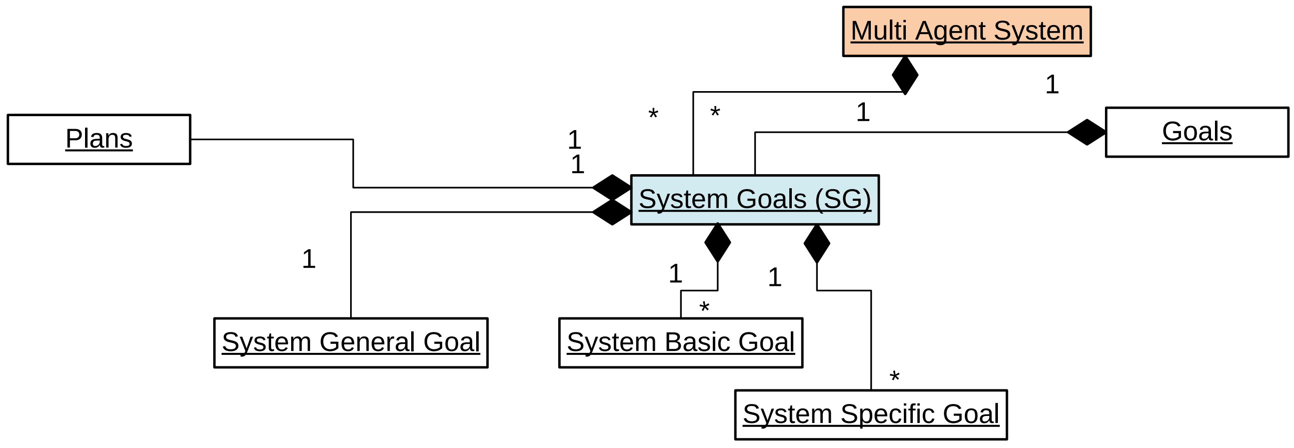


Fig. 4.3: System Goals

4.2. Business Processes. Business Processes are derived directly from the system goals and they describe and identify the steps needed to achieve one or more of the system goals, these steps forming the system plans. For each **Specific System Goal** there is at least one Specific Business Process. And for each Basic System Goal there is one Basic Business Process, that is a collection of a number of **Activities**. Each different sequence of activities that lead to completion of one business process is forming a system plan. Each system plan could be executed by one or more system participant. A collection of these activities into a system plan, to be executed by a system participant, form one system participant plan. Figure 4.4 shows all different types of Business Processes and their relations both the System Plans as well as the System Participants and Table 4.2 summarizes the definitions of all concepts supporting Business Process concept.

4.3. System Participants. System Participants are those system components that are responsible for the execution of plans in order to achieve the system goals. As noted earlier, we expand the definition of MAS to include not only software agents and services but also human actors. The system participants that take the initiative for the achievement of the system goals are called *proactive system participants*, whereas software services, that only respond when they receive a request are called *reactive system participants*. As shown in Figure 4.5, each system participant normally has one or more system plans, and they execute their plans in form

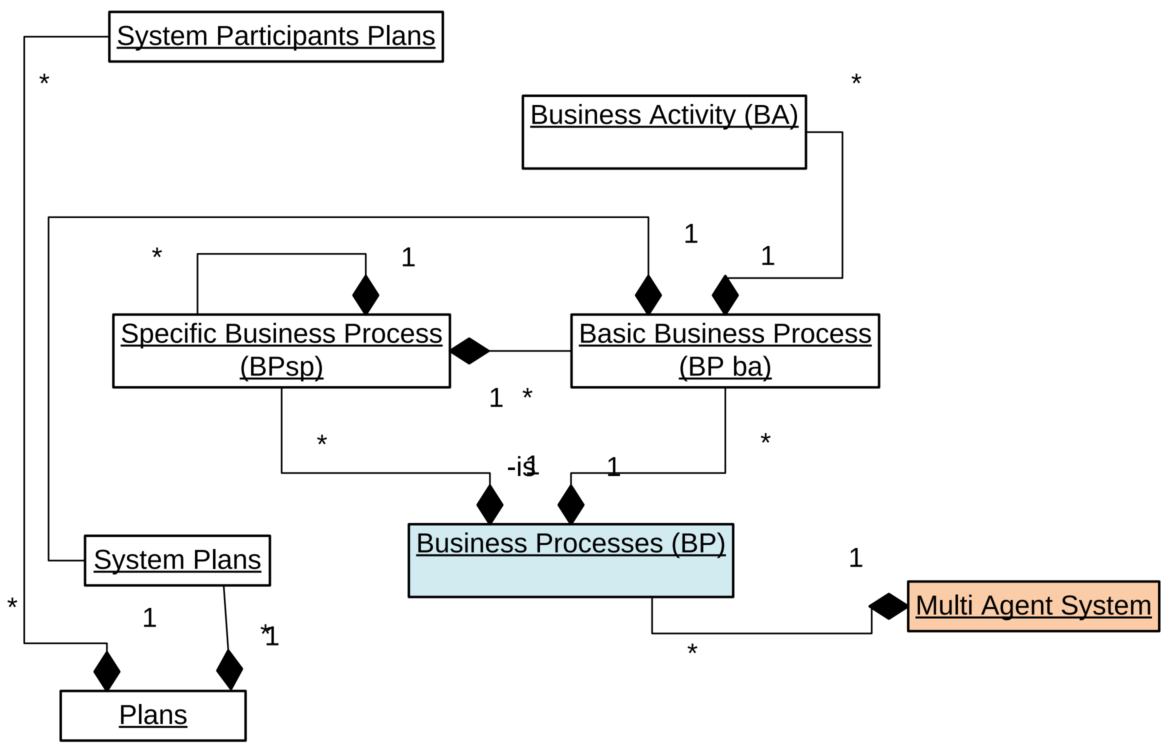

System Participants Plans
Business Activity (BA)
Specific Business Process (BPsp)
Basic Business Process (BP ba)
Business Processes (BP)
Multi Agent System
System Plans
Plans
-is
1
*

Fig. 4.4: Business Processes

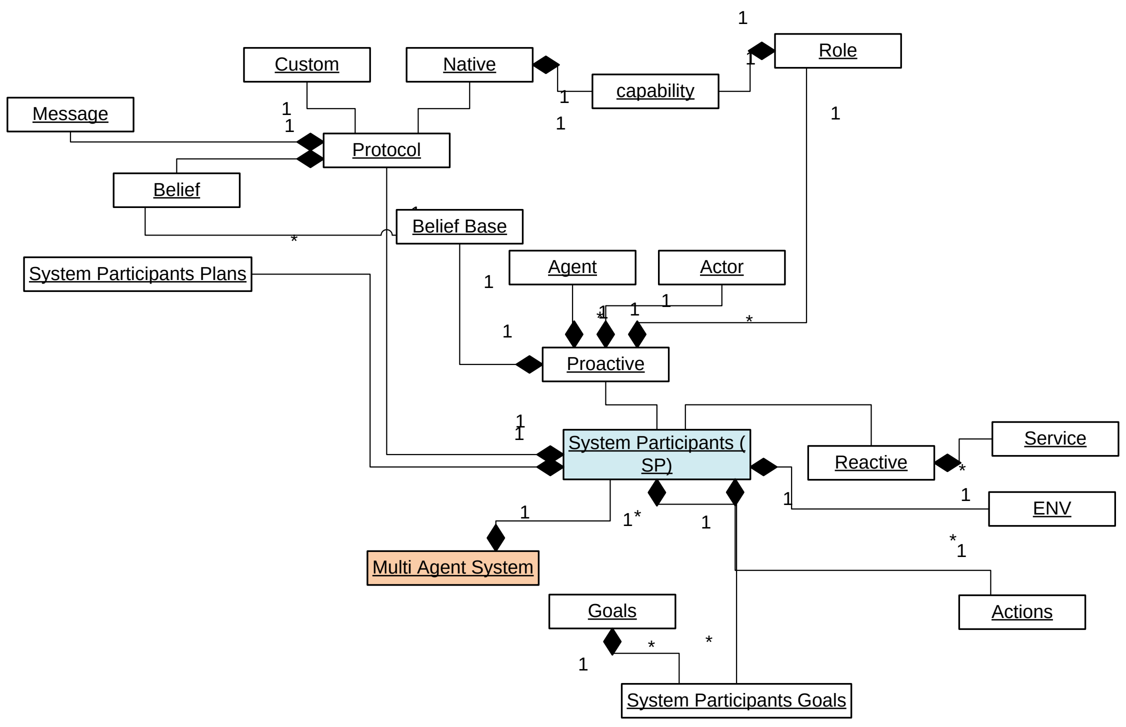


Fig. 4.5: System Participants

Concept	Definition
Goal	A desired state that the system or one or more of its participants aim individually or collectively at reaching
System General Goal	The most general reason for building the system, the achievement of all system goals leads to the achievement of the general goal
System Specific Goal	A functional goal that can be achieved by one or more business processes, it must have one or more sub-goals either Specific or Basic
System Basic Goal	A sub-goal of a system specific goal
Plan	An ordered list of primitive actions, that if executed successfully, lead to the achievement of a goal. A plan normally has preconditions, and the successful execution leads to change of the system state described as a post-condition

Table 4.1: **System Goals Concepts**

Concept	Definition
Business Process	A collection of sub-processes or activities that lead to the achievement of a system goal
Specific Business Process	A collection of sub-processes or activities such that a successful execution of part or all of them leads to the achievement of a specific system goal
Basic Business Process	A collection of activities such that a successful execution of part or all of them leads to the achievement of a basic system goal
Business Activity	A primitive course of actions that involves one or more system participants and may have a precondition.
System Plan	An ordered list of either sub-processes or activities or mix of both if executed successfully, leads to the achievement of a system goal.

Table 4.2: **Business Processes Concepts**

of a series of actions to achieve one of more of their system participants goals. System participants actions do affect the state of the system, the internal state of the system participant itself or both, and they are important for the purpose of monitoring the overall state of the system and to discover any violation of the system norms that are associated with the role this system participant is performing.

Proactive system participants have a belief base, which is a set of facts about themselves, their environment, and other system participants. Each proactive system participant has a set of capabilities and typically it cannot perform a specific role without having the capabilities required for this role. Proactive system participants perform multiple roles during the course of system execution, however their actions need not violate the system norms associated with such roles. System participants communicate using one or more **Communication Protocols**, which are sets of rules determining the format and transmission of a sequence of data in the form of messages between two or more system participants. MSMAS offers a number of pre-defined (**Native Protocols**), as well as allowing the user to define (**Custom Protocols**). The communication protocol can have any number of messages of either of two types: (***Inform Message*** and ***Request Message***), being the basic performatives defined by FIPA [12].

4.4. Institution Structure. Institutions form integral part of the metamodel to support the e-organization structure. Each institution comprises a set of norms that are used to classify agents' actions as norm-compliant or not, of which the latter may result in punishment, depending on what enforcement mechanisms are deployed. An institution itself is a like class or template, which needs to be instantiated before use in order to fill in the identities of the agents that it governs. In particular, and as shown in Figure 4.6 the connection between the rules and the agents is established by the roles that agents play, while the rules themselves are expressed in terms of roles. Some, but not necessarily all, of the actions of an agent will relate to an institution in the sense that an agent's observable action may count as [15] institutional actions, if that agent has the requisite institutional power and if that agent has permission for that (institutional) action at the time it occurs. The consequence of absence of institutional power is simply that the action has no institutional effect.

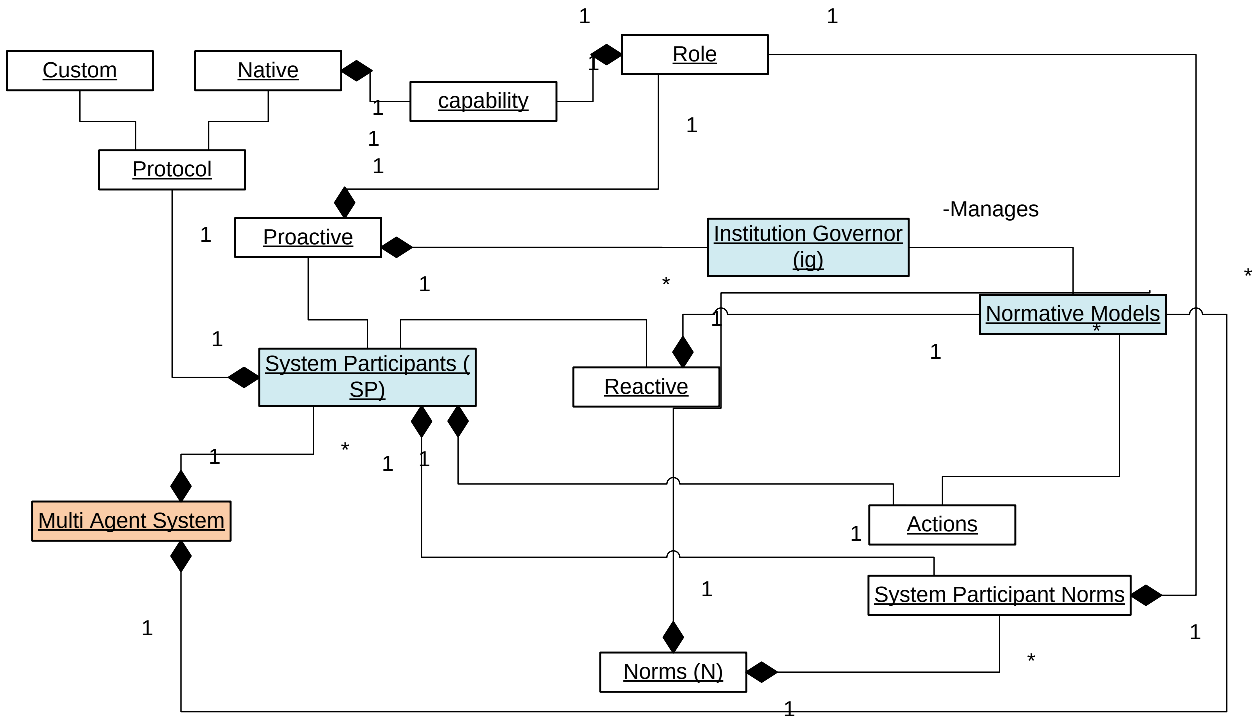


Fig. 4.6: Institution Concepts

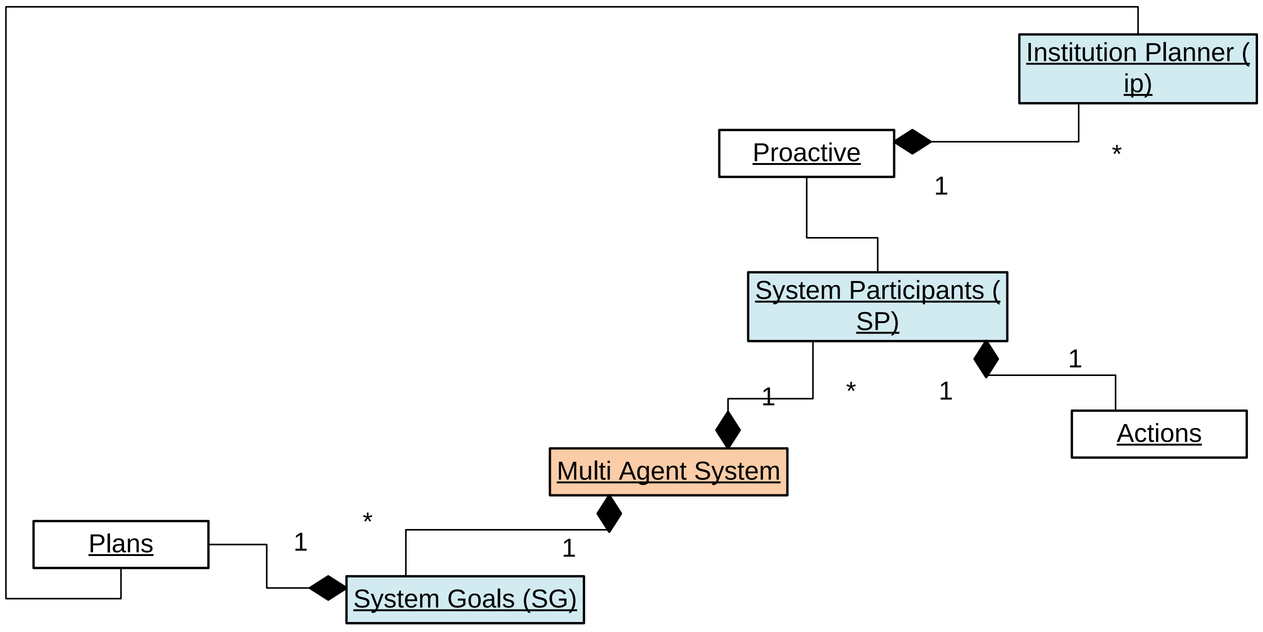


Fig. 4.7: Dynamic Planning

The consequence of absence of permission is that a violation occurs – the action was not norm compliant – and if the violation is observed by an agent with the power to carry out some enforcement action, then there may be consequences for the offending agent. A more detailed discussion can be found in [8].

We observe that there may be: (i) many instances of the same institution, as explained above, for example, each contract [5] can be thought of as being parameterized with the parties to the contract, and (ii) many

Concept	Definition
Reactive System Participant	A software component that works in a stimulus-response manner: they can only respond when triggered by receiving a request.
Proactive System Participant	An autonomous software component that has knowledge of itself, environment and other components and actively uses this knowledge to reason and form plans that lead to achievements of its goals
System Participants Goal	An internal goal that motivates the system participant's internal planning
Service	A reactive system participant, that has predefined set of functions other system participants can use on demand
Agent	A software proactive system participant that actively assesses its internal state and internally plans and acts to achieve its goals
Actor	A human proactive system participant that actively assesses its internal state and internally plans and acts to achieve its goals
Belief Base	A store of all facts (beliefs) that system participant holds about itself, its environment, or other system participants
Role	A specification of a behaviour pattern that the system participants should follow to carry out the function of such role
Capability	The ability of a proactive system participant to perform set of functions specified within a certain role
Protocol	A set of rules determining the format and transmission of a sequence of data in the form of messages between two or more system participants
Custom Protocol	User defined communication protocol
Native Protocol	Predefined communication protocol
Belief	A fact in the form of element in the state of a system participant, environment, or both

Table 4.3: **System Participants Concepts**

different institutions, comprising different sets of norms, serving different purposes and that it is quite likely that an agent will be subject to the governance of more than one institution at the same time. Thus, the primary purpose of the institutional component of the metamodel is to provide a mechanism whereby different sets of constraints may apply to agent behaviour at different times, so that the behaviour of active entities that make the system work can be indirectly influenced without re-coding and re-starting.

4.5. Self-managing through Dynamic Planning. Planning in its simplest form is the process of finding a sequence of actions that leads to the achievement of a goal. We use "dynamic planning" for the on-demand process of finding all possible sequences of actions that lead to the achievement of a defined goal (state) based on the current state of the system and resource availability. The purpose of dynamic planning is to enable the system to overcome the failure of one or more of its components, thus realizing an aspect of self-management. Such behaviour is essential for designing business systems that are able to progress flexibly with their functions without the need to follow precisely a predefined course of actions. We are working on the definition of details of the dynamic planning aspect and our current and future research activities include the answering of questions such as: what is the best planning language to use? What is it realistic to expect dynamic planning to cover? Are the system agents self-contained, with full internal reasoning capability or should the planning service be offered for both global planning and system participants' planning?

Although some questions are not fully resolved in this aspect of the system, we have chosen to support self-management by making the following components – as shown in Figure 4.7 – an integral part of the metamodel: an institution planner, that is one or more software agents that are responsible for re-planning on demand for other system participants based on the available system resources. The current design of the metamodel supports our dynamic planning view and is easily extensible to include more detailed concepts once the remaining questions are answered.

5. Discussion and Future Work. We have described briefly the key features of the MSMAS methodology, which aims to (i) overcome the issues we have found with other MAS development methodologies and (ii) attract a wider range of users to adapt MAS concepts in business applications. MSMAS has clear and well-defined steps that should enable developers to design any small scale MAS and makes MAS concepts accessible and easy to comprehend by business users as well as academics. The methodology is supported by a metamodel, presented here, that covers most common MAS concepts and supports formal system description for verification and implementation purposes. Our aim has not been to define a new MAS metamodel that fits with all methodologies or unifies all other metamodels, but rather to support our methodology with a small, well-defined set of concepts that could be easily understood and specified in a business context without compromise on the most common agreed-on MAS concepts and structure. Our metamodel can be mapped to other metamodels, and could be used as a base for generating code.

The most important features supported by MSMAS are (i) formal specification using DecSerFlow notation, (ii) support for organizational view through the explicit definition of e-institutions, and (iii) the technical support of self-management through dynamic planning and usage of norms.

Two particular features of our methodology and its metamodel that stand out, are (i) the inclusion of an institution structure to support the organizational view of a MAS, and (ii) the use of dynamic planning to allow for self-management and thereby the opportunity for a higher degree of flexibility.

We are currently developing tools to support all the phases of MSMAS and a critical next step is a full evaluation of MSMAS in practice. Other planned future work include the establishment of the most appropriate means for specifying the system norms, describing system and agent plans to support dynamic planning, and deployment methods for distributed MAS systems.

REFERENCES

[1] C. Atkinson and T. Kühne, *Model-Driven development: A metamodeling foundation*, IEEE Software, 20 (2003), pp. 36–41.

[2] G. Beydoun, G. C. Low, B. Henderson-Sellers, H. Mouratidis, J. J. Gómez-Sanz, J. Pavón, and C. Gonzalez-Perez, *Faml: A generic metamodel for mas development.*, IEEE Trans. Software Eng., 35 (2009), pp. 841–863.

[3] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini, *TROPOS: An agent-oriented software development methodology*, Autonomous Agents and Multi-Agent Systems, 8 (2004), pp. 203–236.

[4] M. Dastani, *Programming multi-agent systems.*, in CLIMA, M. Fisher, F. Sadri, and M. Thielscher, eds., vol. 5405 of Lecture Notes in Computer Science, Springer, 2008, pp. 13–16.

[5] M. De Vos, J. Padget, and K. Satoh, *Legal modelling and reasoning using institutions*, in Proceedings of JURISIN 2010, S. Tojo, ed., vol. 6797 of LNCS, Springer, 2011.

[6] S. A. DeLoach, M. F. Wood, and C. H. Sparkman, *Multiagent systems engineering.*, Int. Journal of Software Engineering and Knowledge Engineering, 11 (2001), pp. 231–258.

[7] F. Dignum, D. Morley, E. Sonenberg, and L. Cavedon, *Towards socially sophisticated bdi agents*, Multi-Agent Systems, International Conference on, 0 (2000), p. 0111.

[8] V. Dignum and J. Padget, *Multiagent organizations*, in Multiagent Systems, G. Weiss, ed., MIT Press, 2^{nd} ed., 2012. In press.

[9] S. Edmunson, R. Botterbusch, and T. Bigelow, *Application of system modeling to the development of complex systems*, in Digital Avionics Systems Conference, 1992. Proceedings., IEEE/AIAA 11th, oct 1992, pp. 138 –142.

[10] E. E.-D. El-Akehal and J. A. Padget, *Pan-supplier stock control in a virtual warehouse*, in AAMAS (Industry Track), M. Berger, B. Burg, and S. Nishiyama, eds., IFAAMAS, 2008, pp. 11–18.

[11] M. Esteva, D. de la Cruz, B. Rosell, J. L. Arcos, J. A. Rodríguez-Aguilar, and G. Cuní, *Engineering open multi-agent systems as electronic institutions*, in AAAI, D. L. McGuinness and G. Ferguson, eds., AAAI Press / The MIT Press, 2004, pp. 1010–1011.

[12] FIPA TC, *FIPA ACL Message Structure Specification*, Tech. Rep. SC00061G, Foundation for Intelligent and Physical Agents, 2003. Available via `http://www.fipa.org/specs/fipa00061/SC00061G.pdf`, retrieved 20120610.

[13] C. Hahn, C. Madrigal-Mora, and K. Fischer, *A platform-independent metamodel for multiagent systems.*, Autonomous Agents and Multi-Agent Systems, 18 (2009), pp. 239–266.

[14] N. R. Jennings, *Commitments and conventions: The foundation of coordination in multi-agent systems*, The Knowledge Engineering Review, 8 (1993), pp. 223–250.

[15] A. J. I. Jones and M. J. Sergot, *A formal characterisation of institutionalised power.*, Logic Journal of the IGPL, 4 (1996), pp. 427–443.

[16] M. Luck, N. Griffiths, and M. d'Inverno, *From agent theory to agent construction: A case study.*, in ATAL, J. P. Müller, M. Wooldridge, and N. R. Jennings, eds., vol. 1193 of Lecture Notes in Computer Science, Springer, 1996, pp. 49–63.

[17] M. Luck, et al., *Agent technology: Enabling next generation computing a roadmap for agent-based computing*, 2008. `http://www.agentlink.org/roadmap/al3rm.pdf`, retrieved 20120322.

[18] L. Padgham and M. Winikoff, *Prometheus: a methodology for developing intelligent agents.*, in AAMAS, ACM, 2002, pp. 37–38.

[19] J. Sudeikat, L. Braubach, A. Pokahr, and W. Lamersdorf, *Evaluation of agent - oriented software methodologies - examination of the gap between modeling and platform*, in Agent-Oriented Software Engineering V, Fifth International

Workshop AOSE 2004, P. Giorgini, J. P. Mller, and J. Odell, eds., Springer Verlag, 7 2004, pp. 126–141.

[20] W. van der Aalst and M. Pesic, *Decserflow: Towards a truly declarative service flow language*, in The Role of Business Processes in Service Oriented Architectures, F. Leymann, W. Reisig, S. R. Thatte, and W. van der Aalst, eds., no. 06291 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2006, Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[21] M. B. van Riemsdijk, K. V. Hindriks, and C. Jonker, *Programming organization-aware agents.*, in ESAW, H. Aldewereld, V. Dignum, and G. Picard, eds., vol. 5881 of LNCS, Springer, 2009, pp. 98–112.

[22] N. J. E. Wijngaards, M. Kempen, A. Smit, and K. Nieuwenhuis, *Towards sustained team effectiveness*, in AAMAS Workshops, O. Boissier, J. A. Padget, V. Dignum, G. Lindemann, E. T. Matson, S. Ossowski, J. S. Sichman, and J. Vázquez-Salceda, eds., vol. 3913 of Lecture Notes in Computer Science, Springer, 2005, pp. 35–47.

[23] M. Wood and S. A. Deloach, *An overview of the multiagent systems engineering methodology*, in The First International Workshop on Agent-Oriented software Engineering (AOSE-2000, 2000, pp. 207–222.

[24] M. Wooldridge, N. R. Jennings, and D. Kinny, *The Gaia methodology for agent-oriented analysis and design.*, Autonomous Agents and Multi-Agent Systems, 3 (2000), pp. 285–312.

[25] M. J. Wooldridge, *On the Logical Modelling of Computational Multi-Agent Systems*, PhD thesis, UMIST, Department of Computation, Manchester, UK, 1992.

[26] F. Zambonelli, N. R. Jennings, and M. Wooldridge, *Developing multiagent systems: The Gaia methodology*, 2003.

Edited by: Marcin Paprzycki
Received: May 1, 2012
Accepted: June 15, 2012

Scalable Computing: Practice and Experience
Volume 13, Number 2, pp. 139–147. http://www.scpe.org
ISSN 1895-1767
© 2012 SCPE

A COGNITIVE MANAGEMENT FRAMEWORK TO SUPPORT EXPLOITATION OF THE FUTURE INTERNET OF THINGS

GIANMARCO BALDINI[*], RANGA RAO VENKATESHA PRASAD[†], ABDUR RAHIM BISWAS[‡], KLAUS MOESSNER[§], MATTI ETELAPERA[¶], JUHA-PEKKA SOININEN[‖], DSEPTIMIU-COSMIN NECHIFOR[**], VERA STAVROULAKI [††] AND PANAGIOTIS VLACHEAS[‡‡]

Abstract. In this article, a cognitive management framework is proposed for ensuring exploitation of the Future Internet of Things (FIoT). Cognitive systems offer self-x and learning. A cognitive system has the ability to dynamically select its behavior through self-management/awareness functionality, taking into account information and knowledge on the context of the operations as well as policies and including the generation of the context itself. The framework is based on the principle that any real world object and any digital object that is available, accessible, observable or controllable can have a virtual representation in the Future Internet, which is called Virtual Object (VO). Basic VOs can be composed in a more sophisticated way by forming Composite VOs (CVOs), which provide services to high-level applications and end-users. The described paradigm is applied to various applications scenarios: smart home, smart office, smart city and smart business. This paper presents some background in IoT, identifies the requirements and challenges, and sets the directions that should be followed.

Key words: Cognitive Systems, Internet of Things, Virtual Objects, Wireless Communications

1. Introduction.

The "7 trillion devices for 7 billion people" paradigm [1], yields that the handling of the amount of objects that will be part of the Internet of Things (IoT) requires suitable architecture and technological foundations. The Internet-connected sensors, actuators and other types of smart devices and objects need a suitable communication infrastructure. While several research projects (e.g., IoT-A [2], CASAGRAS2 [3]) have set out to define architectures or reference models to ensure interactions and facilitate information exchange, there is a significant lack in terms of management functionality and means to overcome the technological heterogeneity and complexity of the pervasive networks in terms of exploitation effectiveness. This is essential for the IoT, in order to enhance context-awareness (by being able to match continuously an evolutionary demands of client applications against an unreliable connection and representation quality of real world objects), provide high reliability (through the ability to use heterogeneous objects in a complementary manner for reliable service provision), energy-efficiency (through the selection of the most efficient and suitable objects from the set of heterogeneous ones, and, in general, through the optimal management of a large population of resource constrained devices) and security in these distributed networks of cooperating objects. The sheer numbers of objects and devices that have to be handled and the variety of networking and communication technologies, as well as administrative boundaries that have to be supported do require a different management approach. The idea is to enable seamless and interoperable connectivity amongst heterogeneous number of devices and systems, hide their complexity to the user while providing sophisticated services and applications [4].

In response to the requirement of overcoming technological heterogeneity this paper proposes a cognitive management framework. The proposed framework aims to provide the means to realize the principle that any real world object and any digital object, which is available, accessible, observable or controllable, can have a virtual representation in the IoT. This means that the functionality or features offered by any kind of object can become part of composite functionality/features, which will be reusable in the context of sophisticated application/service provision in the IoT.

Moreover, the aim of the framework is to provide the foundations, architecture and functionality for a cognitive support paradigm for the IoT, for continuous sensing of client applications, own environment and real world variations. A cognitive system has the ability to dynamically select its behavior (managed systems configuration), through self-management/awareness functionality, taking into account information and knowledge (obtained through machine learning) on the context of operation (e.g., internal status and status of

[*]JRC, Belgium (gianmarco.baldini@jrc.ec.europa.eu).
[†]TU Delft, The Nederlands (R.R.VenkateshaPrasad@tudelft.nl).
[‡]Create Net, Italy (abdur.rahim@create-net.org).
[§]University of Surrey, UK (K.Moessner@surrey.ac.uk).
[¶]VTT, Finland (Matti.Etelapera@vtt.fi).
[‖]VTT, Finland (Juha-Pekka.Soininen@vtt.fi).
[**]Siemens CT, Romania (septimiu.nechifor@siemens.com).
[††]UPRC, Greece (veras@unipi.gr).
[‡‡]UPRC, Greece (panvlah@unipi.gr).

environment), as well as policies (designating objectives, constraints, rules, etc.).

In the light of the above, cognitive technologies constitute an efficient approach for addressing the technological heterogeneity and obtaining context awareness, reliability and energy efficiency. Cognitive technologies have been applied to the management of diverse heterogeneous technologies (e.g., wireless access, backhaul/core segments). The proposed framework applies this successful paradigm for solving problems that are particular to the IoT. Therefore, new IoT-oriented cognitive functionality will be provided, which will be part of the service layer of the Future Internet. A cognitive system consists of the cognitive engine (offering intelligence and service capabilities) and the reconfigurable/managed part, which is technology specific. The engine interacts with the managed part and with other engines. Each managed part is directly controllable by one engine (i.e., other engines have to interact with the managing cognitive engine in order to affect a specific managed resource). Through this approach there is the accomplishment of the abstraction of the technological heterogeneity, which leads to the removal of the sector specific boundaries. Of course, the realization of above described cognitive capabilities relies intensively on the support of autonomic capabilities on both thing level and support platform level. Context awareness is inherent in the model, while policies and decision-making (part of autonomic features) can be oriented to address the targets of enhanced reliability and energy-efficiency.

Additionally, the proposed framework addresses security, resilience and user privacy issues, which are vital for the Future Internet, though a policy management approach where access to data and resources is regulated by policies and access levels (also common with current practices in autonomic computing and networking).

From the users/applications perspective, three concepts - IoT, ubiquitous computing, and ambient intelligence - aim at delivering smart services (where smartness reflects the accuracy of context sensing, service matching, platform use and time wise capability) to users [5]. A part of the smartness relies on context awareness, e.g., service provision according to the needs that exist at the place, time and overall situation. This is not all. At a societal level, smartness also requires that the needs of diverse users and stakeholders are taken into account both design time (templates) and run-time (learning capability). Stakeholders can be the owners of the objects and of the communication means. Different stakeholders that are part of the current Internet milieu and they will be part of Future Internet, have interests that may be adverse to each other and their very own criteria on how objects could be used and should be accessed. Clark et al. [6], calls this process the tussle and any Future Internet framework should be able to accommodate such tussle to support a smooth evolution of Future Internet of Things, as expected to be a world of competing and conflicting contexts and demands. So a key challenge that needs to be tackled includes the handling of the diversity of information while respecting the business integrity, the needs and rights of users and of the various stakeholders.

The approach presented in this paper aims to overcome the issues above by bringing further intelligence in the Internet of Things. The remaining part of the paper is organized as follows: Section II describes the proposed approach to address such challenges through a cognitive management framework based on the concept of virtual object. Section III describes the Security and Privacy aspects. Section IV describes the application of the cognitive management framework to two scenarios: smart home and smart office. Section V concludes the paper and provides future directions in this research area.

2. Cognitive Management Framework for Future Internet of Things.

2.1. The framework. The proposed framework is targeted to concealing technological heterogeneity and for satisfying the requirements of different users/stakeholders so as to meet the objectives for context awareness, reliability, and energy efficiency. Additionally, security will be a primary concern and an important property at all levels of the cognitive framework. The framework comprises three main levels of enablers, which are reusable to various-diverse applications.

In each level there are scalable fabrics, which offer mechanisms for the registration, look-up and discovery of entities, and the composition of services.

Cognitive entities at all levels provide the means for self-management (configuration, healing, optimization, protection) and learning. In this respect, they are capable of perceiving and reasoning on their context (e.g., based on event filtering, pattern recognition, machine learning), and of conducting associated knowledge-based decision-making (through associated optimization algorithms and machine learning).

Through such features the proposed framework constitutes an open networked architecture encompassing highly intelligent (i.e., adaptive, knowledge based, eventually proactive, etc.) software.

The virtual objects (VOs) are primarily targeted to the abstraction of technological heterogeneity. VOs accomplish their role through the cognitive management and handling of real-world or digital objects (e.g.,

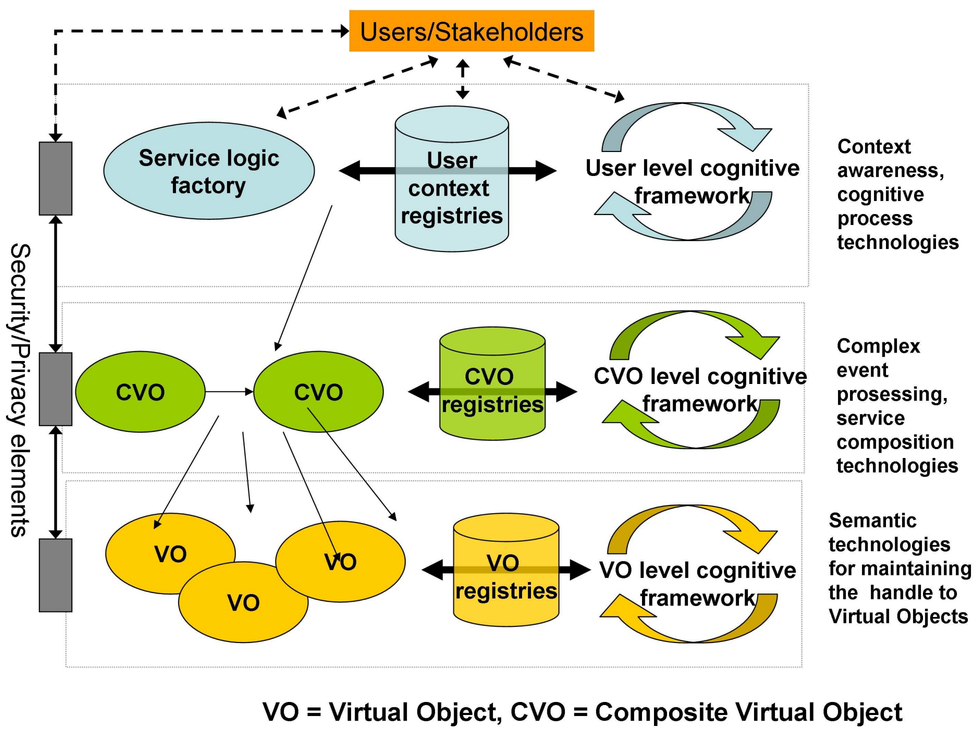

Users/Stakeholders
Service logic factory
User context registries
User level cognitive framework
Context awareness, cognitive process technologies
Security/Privacy elements
CVO
CVO
CVO registries
CVO level cognitive framework
Complex event prosessing, service composition technologies
VO
VO
VO
VO registries
VO level cognitive framework
Semantic technologies for maintaining the handle to Virtual Objects
VO = Virtual Object, CVO = Composite Virtual Object

Fig. 2.1: Layers of the Cognitive Management framework

sensors, actuators, devices, etc.). VOs are cognitive virtual representations of real-world objects and/or digital objects.

User/stakeholder related objects will convey the respective requirements. The entities will be capable of detecting human intentions and behavior, inferring, and eventually acting on behalf of the users. In this respect, there is seamless support to users, which is in full alignment with their requirements (the learning capabilities of the cognitive entities of this layer will be applied for acquiring knowledge on user/stakeholder preferences, etc.). Capabilities for governing the entities will also be included (through any type of interaction - multi-modal interactions)

Composite virtual objects (CVOs) will be using the services of virtual objects. A CVO is a cognitive mash-up of semantically interoperable VOs that renders services in accordance with the user/stakeholder perspectives and the application requirements.

The concept of VOs is not new. Object-oriented (OO) approaches have been used in computer programming for decades and distributed objects are used in Object-oriented middleware applications in the Web 7. The intention is not to create new digital representations/objects, but to combine previous concepts with cognitive management mechanisms to create and maintain dynamic, intelligent virtual representation of real world/digital objects that can enhance the Future Internet.

As already introduced, the framework comprises three layers of cognitive components, which are depicted in Figure 2.1.

The first cognitive management layer (*VO level cognitive framework*) is responsible for managing the VOs and for the abstraction of the technological heterogeneity. Real-world or digital objects (e.g., sensors, actuators, devices, etc.) are represented in the first layer through VOs. In current practice and standardization the most prevalent solution is considered the use of RESTful services, based on experienced capabilities of things. The management layer is responsible for the VO lifecycle (i.e., creation, update, destruction) and to address the heterogeneity by defining the logical links among VOs. For example the container transported by a truck is a VO as the truck itself. A tracking device on the truck (with GPS and communication terminal) is also a VO.

The second cognitive management layer (*CVO level cognitive framework*) is responsible for composing the VOs in Composite VO (CVO). CVOs will be using the services of VO to compose more sophisticated objects.

A CVO is a cognitive mash-up of semantically interoperable VOs that renders services in accordance with the user/stakeholder perspectives and the application requirements. For example, the combination of the trucks, the transported goods and the tracking device is represented in the cognitive framework as a CVO.

The third level (*User level cognitive framework*) is responsible for interaction with User/stakeholders. The cognitive management frameworks will record the users needs and requirements (e.g., human intentions) by collecting and analyzing the user profiles, stakeholders contracts (e.g., Service Level Agreements) and eventually acting on behalf of the users. In this respect, there is seamless support to users, which is in full alignment with their requirements (the learning capabilities of the cognitive entities of this layer will be applied for acquiring knowledge on user/stakeholder preferences, etc.). Capabilities for governing the entities will also be included (through any type of interaction - multi-modal interactions).

An alternative representation of the framework is described in Figure 2.2, where the lowest level is connectivity level and it is composed of real world objects. These objects may or may not be communication enabled. Then we define the VOs and CVOs as above. These virtual objects have cognitive functionalities depending on their capability sets. The right composition of abstracted virtual objects is done at this level (i.e., VO level). The topmost level is the service level (may also be called as User level since users could also influence the way services are configured). Based on the required service the CVO are composed on the fly. The user requirements, the context and the analysis of the available resources contribute to the dynamic service composition and orchestration to offer the right services at the right time to the user.

The next section describes more in detail the lifecycle of the VO/CVOs and the dynamic composition of services and applications.

2.2. Virtual object lifecycle and dynamic composition of services. Since they have to represent dynamically changing real world objects, VOs and CVOs should be dynamically created, updated and destructed. More importantly, the services provided by VO and CVO have to be dynamically composed to support the users needs in function of space, time or context [9]. For example: provisioning of communication services and data may be different for Healthcare services during an emergency crisis (i.e., the aftermath of an earthquake) or during routine operations.

The proposed framework encapsulates the support for maintaining and exploitation of VOs and support for transferring the incentives of stakeholders to CVOs. We can say that the framework (and support implementation platform) has to support four levels of cognition. First, the support platform has to keep the continuous link to the real world. Secondly, it has to react to the service context changes of the VO and CVO. For example, the consequences of an earthquake are that VOs representing base stations or routers may be destroyed or provide degraded services (i.e., lower data rates). Thirdly, it must be able to identify, understand, and learn from changes in the context. For example, in normal situations public safety officers may not be allowed to access sensitive information on civilians (i.e., medical conditions) but in emergency crisis, framework must provide the access to these data. Fourthly, it must be able to resolve conflicts between different infrastructures (i.e., set of CVOs) or the tussle described in the introduction. For example, enterprises are always looking for information or resources to undermine the competition from other enterprises. Users would like to use connectivity resources without paying for them (e.g., Skype), while telecom providers would like to maximize the revenues.

The cognitive management framework will also control the lifecycle of the VOs and CVOs: their creation, destruction, and update. While (real world or digital) objects may be owned (controlled) by a particular stakeholder, the VO can be owned (controlled) by particular service providers. And in turn, CVO may be owned (controlled) by yet another provider who adds value by combining different virtual objects and providing these combinations to users. This hierarchical structure leads to a rather complex eco-system, but it opens new opportunities for all stakeholders. Furthermore, the cognitive management system will ensure that the complexity of this eco-system will be well hidden from the different players and stakeholders. The proposed life-cycle and the composition of services and applications are depicted in Figure 2.3.

With reference to Figure 2.3, the following steps describe the lifecycle:

1. A new entity is discovered in the area managed by a specific instance of implementation platform. For example: a PC is switched-on, authenticated and connected to the network. The PC is registered in the VO registry of instance with the related features.
2. In a similar way, CVOs are created by analysis of the VO registry or directly created by composite objects in the real domain (e.g., a taxi).
3. At the user level, an instance authenticates a new user and his set of preferences like type of information he is interested in, type of used terminal and so on. The instance periodically records the users context

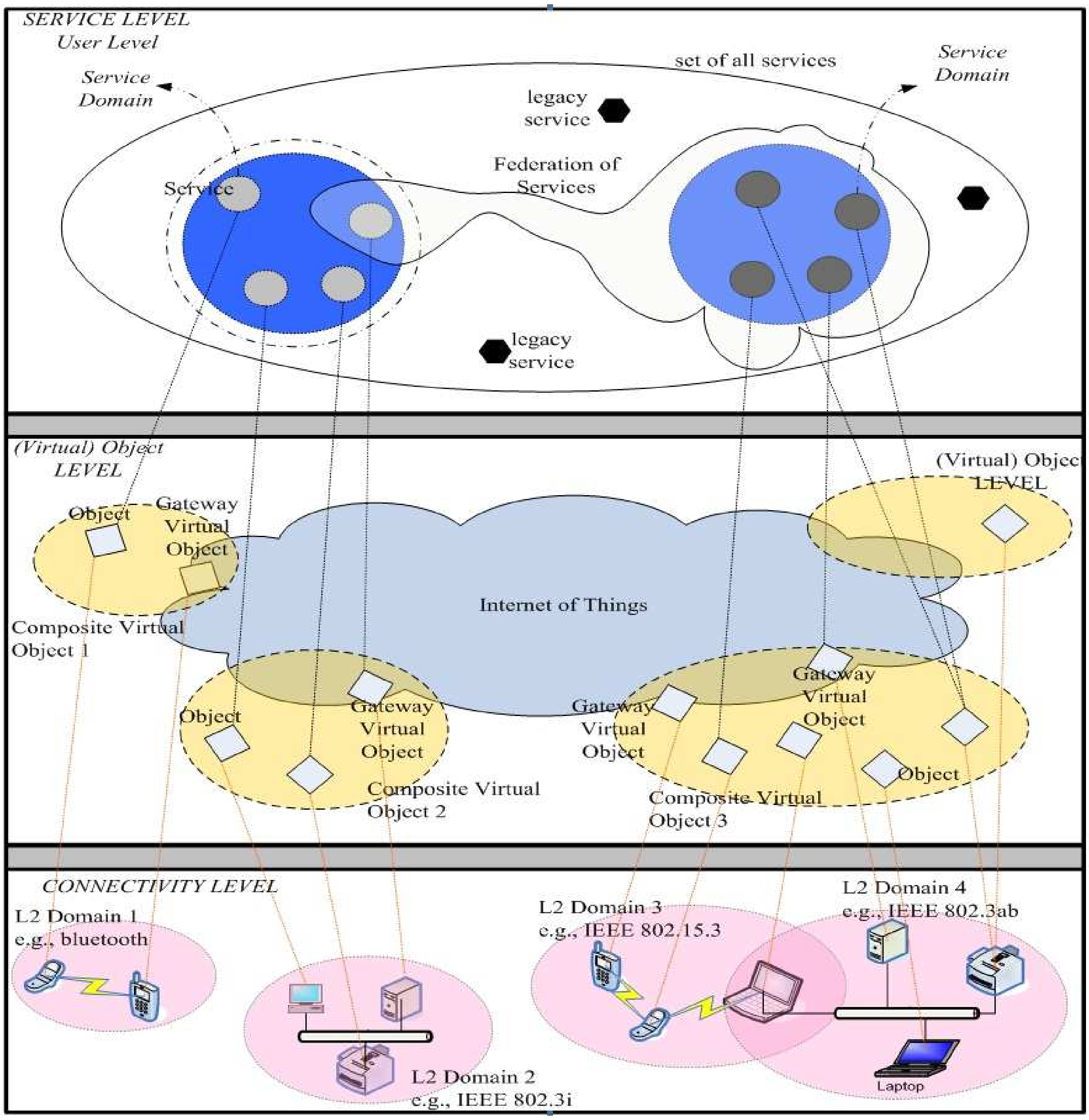


Fig. 2.2: Connect Physical objects, VO and Services

and location.

4. At the service level, the instance correlates the services with the CVO/VO present in the registry. For example, the taxis with credit card payment systems (CVO) present in a specific urban areas, which are available to provide transportation services.
5. Any application can use an instance to access services, information on CVO/VO and users data (see section III for security aspects). For example: a taxi booking application can record the users needs in an area (e.g., need for transportation with credit card) and their location. Then, the application matches the needs with the available CVO (e.g., taxi with credit card services). Specific needs can also be addressed through supported semantic modeling, reasoning or machine learning capabilities. For example: a taxi with a large trunk for suitcases.

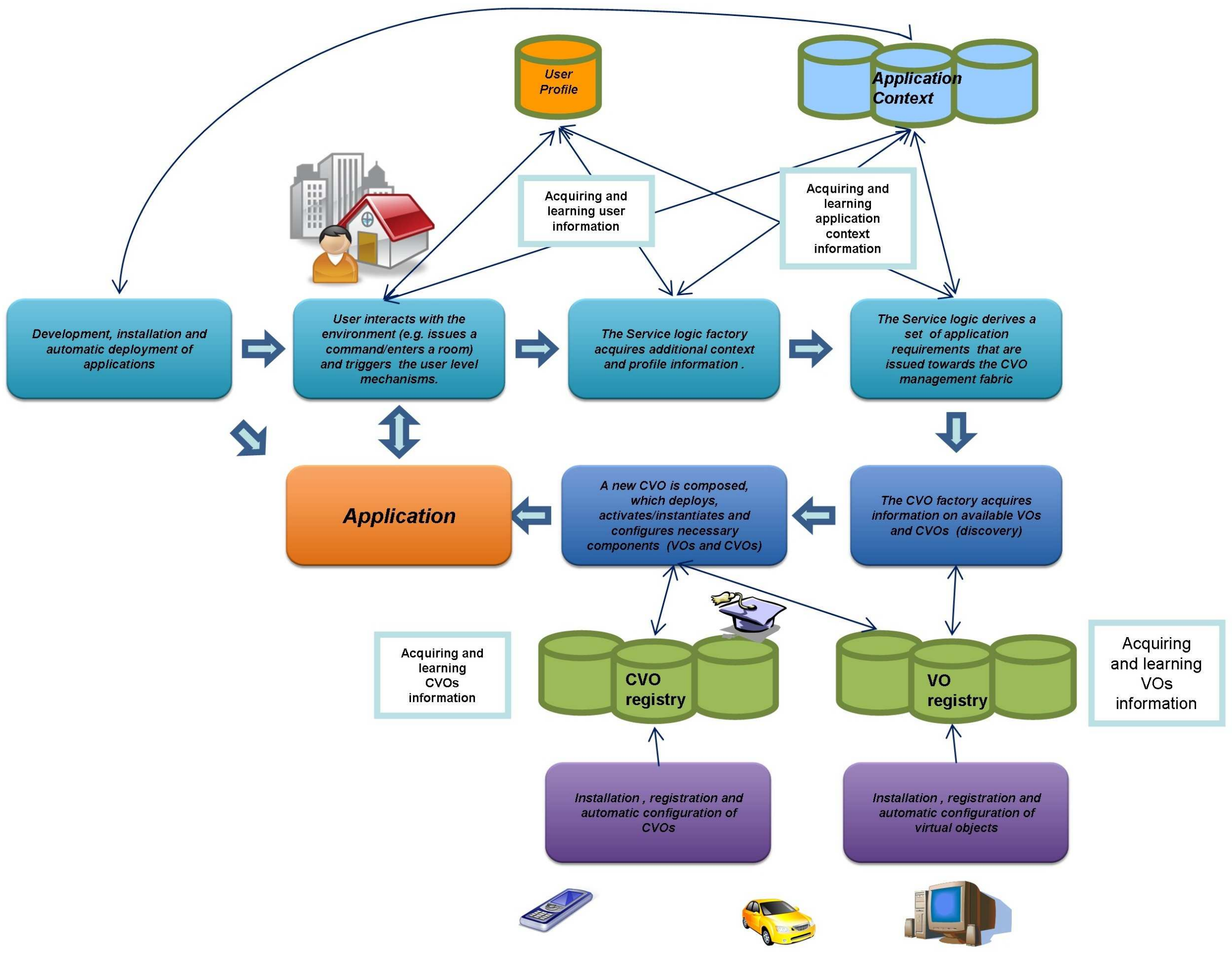

User Profile
Application Context
Acquiring and learning user information
Acquiring and learning application context information
Development, installation and automatic deployment of applications
User interacts with the environment (e.g. issues a command/enters a room) and triggers the user level mechanisms.
The Service logic factory acquires additional context and profile information .
The Service logic derives a set of application requirements that are issued towards the CVO management fabric
Application
A new CVO is composed, which deploys, activates/instantiates and configures necessary components (VOs and CVOs)
The CVO factory acquires information on available VOs and CVOs (discovery)
Acquiring and learning CVOs information
CVO registry
VO registry
Acquiring and learning VOs information
Installation , registration and automatic configuration of CVOs
Installation , registration and automatic configuration of virtual objects

Fig. 2.3: A platform instance life-cycle and services composition

6. An instance can also be used to support the deployment and activation of the application on the users terminals, because records the capabilities of terminals in the CVO/VO registries. It can support a more efficient deployment and tailoring for specific application, beyond the security requirements described in the next section.

3. Security and Privacy Aspects. As described in the previous sections, the proposed framework has to represent (through the VO concept) any type of real object, which may or may not already be connectable to the internet. Real objects could provide privacy sensitive data. In addition, the combination of data from multiple real objects into a new VO could be privacy sensitive. A real object could also provide access to control a particular system (i.e., actuator) for which some kind of access control is required. So the framework has to address the security and privacy issues to enable the dynamic creation of VOs and re-use of real objects and VOs for providing reconfigurable services. The challenge with respect to security and privacy is to integrate novel privacy and security techniques right from the start such that security and privacy would not become an afterthought or add-on feature. Another challenge is to integrate existing legacy systems, which may have proprietary security systems.

The framework is based on the concept that access to data, resources and services represented by VO/CVO must be regulated through a sticky policy management approach [8]. The underlying notion behind Sticky Policy is that the policy applicable to a piece of data travels with it and is enforceable at every point it is used. Users will therefore be able to declare privacy statements defining when, how and to what extent their personal information can be disclosed.

Each VO should contain the following information: 1) the available resources, 2) the access level rights and 3) associated policies valid for this object. This information will be added to the VO in the creation phase, which also includes authentication. For example: a GSM terminal can become a VO, when it is authenticated by the GSM network. In most cases, real-object will not have an authentication mechanism and a security agent must be defined to implement the authentication. In other cases, specific proprietary security systems are already present and a security mediation layer must be defined.

A policy can have multiple rules, which define what resources can be accessed and managed (i.e., used) by a specific access level right. Policies can be defined independent by the existence of associated virtual objects. Note that there are different levels and types of access rights: create, read, modify. Under specific rules, access rights can be delegated: a virtual object can acquire for a specific time, or space or context the access rights of another virtual object. This is particularly useful for the creation of automatic agents. The access level rights can also be used in the ontology and lookup mechanism in the CVO/VO registries: if a virtual object has higher access rights than the entity accessing the dictionary, the virtual object will not appear. Another interesting approach that needs to be considered is claim based access control [10]. In claims based access control access to a resource is provided based on one or more claims. Examples of claims are the proof that "I am over 18 years old", "I am an employee of this company", "I am a guest at this hotel". The enforcement of policies will guarantee privacy of data because data cannot be accessed by entities with the insufficient access right. Data can be protected by additional cryptography services: an entity may have the access rights to access a specific data, but this may be encrypted. Then the entity may also need access to a cryptographic service as well. Policies will also support the concept of Trust Management and reputation scheme. Access rights can be removed or increased by a central authority. For example, an entity may have decreased access rights if there was a security breach. Or, an entity may have increased access rights in occasion of a specific context: for example, public safety officers may have access to sensitive information (e.g., building plans) in case of a crisis management.

Figure 3.1 provides a pictorial view of the security framework and functions. When a VO is authenticated and stored in the VO registry, its access levels and related policies are defined. Users data is also stored as a VO with specific access levels to preserve the privacy of the user. The existing access policies are stored in a distributed policies database, which also records relevant global and local regulations. For example: radio frequency spectrum regulations to regulated use of spectrum in cognitive radio networks or privacy regulation to regulate access to data. The security and policy management functions regulate the access to CVO/VO on the basis of the user access levels and user context. In most cases, existing security frameworks are already in place: for example the authentication functions of GSM/UMTS networks. In these cases, security gateways are required to mediate the necessary information. The main challenge for the implementation of these concepts in the proposed architecture is scalability. A security framework like PKI can handle a specific number of objects/entities but the ICT call requires the management of thousands of objects. Functions like audit and accountability are quite resources consuming if applied to very large networks. A hierarchical approach could be proposed, with hierarchical domains based on geography or contexts. A specific organization can decide to provide only an interface with a subset of VOs to the rest of the Future network.

4. Application Scenarios. This section describes some a set of application scenarios in order to highlight the real life value of the envisioned technology. Cognition and CVOs are cross-domain, and aim to overcome current interfacing efforts. Ranging from home and office domotics use up to industrial infrastructure, the cross-domain aspect generates high volumes of specific developments, often erroneous or inflexible. In computer industry the plug-and-play capabilities have been received as a relevant improvement in terms of functionality and natural use. This fact was based on standardized interfaces and protocols. Now, the avalanche of pervasive computing infrastructure pushes us to raise the level of plug-and-play to very heterogeneous devices and contexts. These problems cannot be solved only with appropriate protocols. Our in-place service infrastructure needs to understand what exactly the service needs to look like. Here is the place where Cognition and CVO are called to help at virtual level.

In the following we present some very simple examples.

Our homes and offices are now intensively populated with devices: surveillance cameras, printers, phones, computers, and a large variety of sensors such as the ones for light and temperature or humidity. We need to or want to share these resources, attach security to them, use and link them (controlled or not) with needs, intentions and processes. We need to give a meaning, real time effectiveness and for this fact we need to implement cognition capabilities. That is the paradigm of smart home.

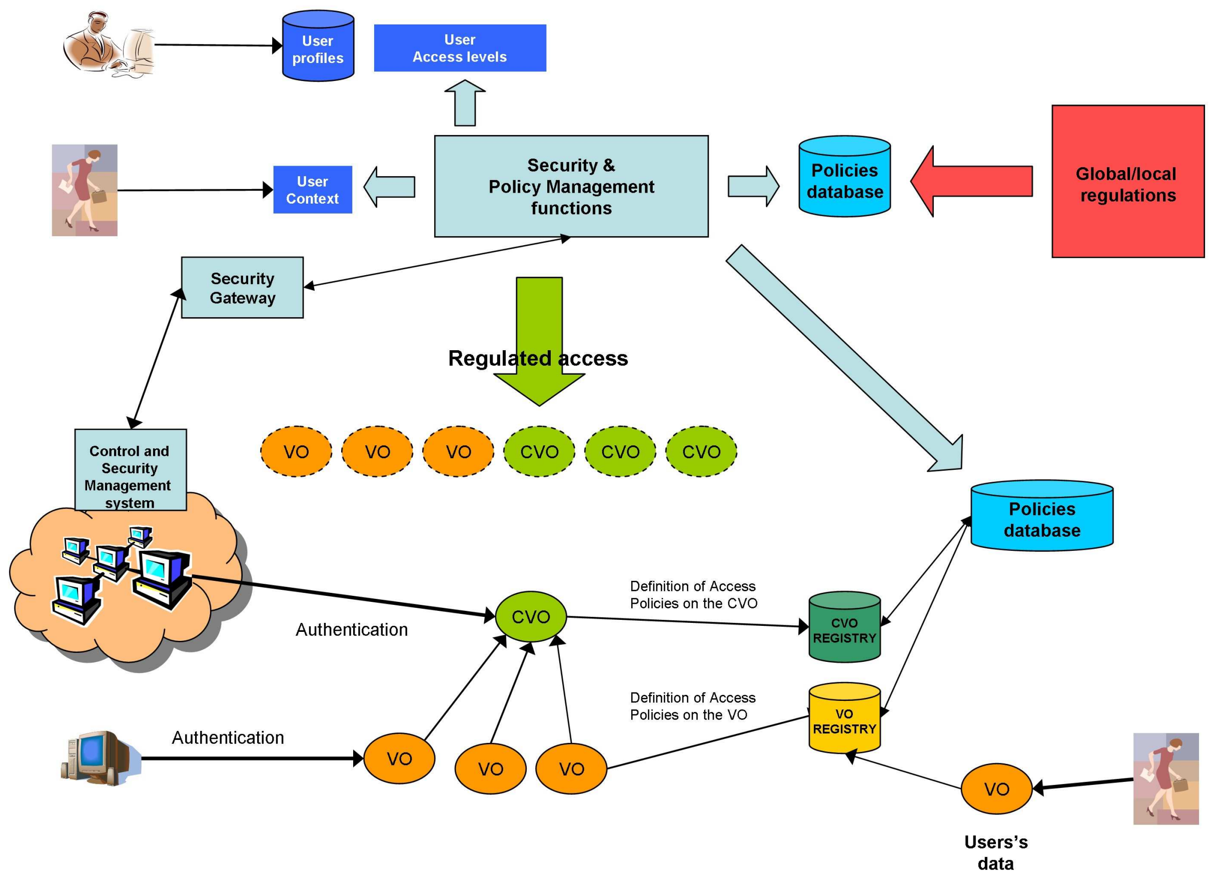


Fig. 3.1: Security framework and functions

Lets imagine that Mary is living with her family, including her father who needs permanent care due to a chronic health issue. Due to her daily duties Mary needs to travel in a neighboring city for some meetings. Her father remains at home, monitored by a number of specialized devices and some control cameras over the usual living space. Mary expects that in case of a possible critical situation, the problem is detected, relevant information is gathered, and specialized medical team and herself alerted in a consistent and timely manner (like a message on her smart phone with a short report attached). In a classical perspective, this kind of particular scenario requires specialized services developed, including spatial understanding of the problem (like the correlation of a body falling in a room with the variation of a body parameter).

Now lets have a look how the problem is tackled using VO/CVO and cognitive infrastructure. A person who needs medical monitoring is equipped with sensors specialized in various measurements for continuous supervision. All these sensors have their own virtual objects representations usable in order to trigger adequate reaction. The same principles apply to an accelerometer or a video camera (see the example of a body falling). Based on these primitive virtual objects, a cognition enabled infrastructure will be able to infer a CVO who refer all relevant virtual objects and compose the relevant services used to solve the case: calling emergency services, family being alerted, etc. The same can be expressed regarding the time dependence between all observed variations. As a consequence, a custom problem is expressed in terms of a set of cognitive enabled strategies and policies. Even more, the cognition doesnt remain fixed but ask for users/actors active feedback and incorporate successful executed steps and experiences. This scenario is not restricted just to a sensing approach, but can involve in the loop actuators. One benefit is the real time inclusion of correlated actuators effects in the cognition correction.

5. Conclusions. This paper described a new cognition based framework approach to manage the complexity, huge amount of data, systems and services which the Future Internet of Things will be comprised of. The approach is based on the concept of CVO and VO to simplify the management of heterogeneous systems and data. Specific features of digital and physical objects can be represented as VO attributes. Security framework and functions regulates the access to CVO/VO through a sticky management policy.

Future developments will investigate how to address scalability of the proposed approach: the proposed architecture must be able to support millions of CVO/VOs, make them accessible from various instances deployed in various domains and offer the necessary mix of autonomic and learning capabilties for optimal matching of demands with offer in an un-reliable, resource access, energy and communication constrained environment.

REFERENCES

[1] M. Uusitalo, *Global Vision for the Future Wireless World from the WWRF*, Vehicular Technology Magazine, IEEE , vol.1, no.2, pp.4-8 (2006)

[2] *IoT-A Website*, Available at http://www.iot-a.eu/

[3] *CASAGRAS2 Website*, Available at http://www.iot-casagras.org/

[4] M. Weiser, *The Computer for the Twenty-First Century*, Scientific American, pp. 94-10, (1991)

[5] J. Schonwalder, M. Fouquet, G. Rodosek, I. Hochstatter, *Future Internet = content + services + management*, IEEE Communications Magazine, , vol.47, no.7, pp.27-33, (2009).

[6] D.D. Clark, J. Wroclawski, K.R. Sollins, R. Braden, *Tussle in cyberspace: defining tomorrow's Internet*, Networking, IEEE/ACM Transactions on , vol.13, no.3, pp. 462- 475, (2005).

[7] S. Vinoski, *Web services interaction models. Current practice,* IEEE Internet Computing, , vol.6, no.3, pp.89-91, (2002).

[8] M.C. Mont, S. Pearson, P. Bramhall, *Towards accountable management of identity and privacy: sticky policies and enforceable tracing services,* Database and Expert Systems Applications, 2003. Proceedings. 14th International Workshop on , vol., no., pp. 377- 382, 1-5 Sept. 2003.

[9] F. Toutain, A. Bouabdallah, R. Zemek, C. Daloz, *Interpersonal context-aware communication services,* IEEE Communications Magazine, , vol.49, no.1, pp.68-74, January 2011.

[10] W.A. Alrodhan, C.J. Mitchell, *Enhancing user authentication in claim-based identity management,* Collaborative Technologies and Systems (CTS), 2010 International Symposium on , vol., no., pp.75-83, 17-21 May 2010

[11] J. Hoebeke, G. Holderbeke, I. Moerman, M. Iacobsson, V. Prasad,. N. I. Cempaka Wangi, I. Niemegeers, S. Heemstra de Groot, *Personal Network Federations*, Proceedings of the IST Summit 2006, Myconos, Greece, June 2006

Edited by: Marcin Paprzycki
Received: May 1, 2012
Accepted: June 15, 2012

Scalable Computing: Practice and Experience
Volume 13, Number 2, pp. 149–156. http://www.scpe.org
ISSN 1895-1767
© 2012 SCPE

AN ALGORITHM FOR GRAVITY ANOMALY INVERSION IN HPC[*]

NEKI FRASHERI[†] AND SALVATORE BUSHATI [‡]

Abstract. In the paper we analyse results from the inversion of geophysical anomalies in high performance computing platforms. We experiment the solution of this ill-posed problem, trying to bypass the complexity of the calculations using simple algorithms that require huge calculation capacities offered by parallel systems. The gravity anomalies are considered because of the simplicity of the gravity modeling in geophysics.

Key words: geophysics, gravity inversion, parallel computing

AMS subject classifications. 86A22, 68N19

1. Introduction. The inversion of geophysical anomalies is a typical ill-posed problem [1]. The inversion process consists of extrapolating from a 2D surface distribution of a physical field to a 3d spatial distribution of physical proprieties, which may led to alternate solutions with divergences in shape and spatial distribution of anomalous bodies (see for example [2]).

Geophysical inversion is studied for a long time and a multitude of methods exist for different contexts. Recently the attention is shifting towards the use of high performance computing (HPC), fueled by the increase of popularity of computer clusters and graphic processing units (GPU) that make parallel data processing widely available. As result a number of publications dealing with the use of parallel computing in geophysical inversion appears scattered in the landscape of complexity of the problem.

Rickwood and Sambridge analyse MPI parallel implementations of the direct search inversion algorithm using the neighbourhood method, replacing the concept of master node with that of iteration in order to achieve the scalability, and evaluating the impact of Amdahl's law for the speedup of software in parallel systems [5].

Loke and Wilkinson used parallel computing in GPU for the 2D smoothness-constrained least squares optimization for the Compare R method of the inversion of resistivity data, and studied the related runtime for models with profiles of only 35 electrode points [6].

Zuzhi et al. used the Simulated Annealing algorithm parallelized with MPI for the inversion of magnetotelluric anomalies, starting with 1D models, using 2D conventional inversion for calculation of resistivity and frequency parallel computation for 2D and 3D forward modeling [7].

Wilson and al. dealt with the massive 3D inversion of airborne gravity gradiometry anomalies based in the single-point Gaussian integration, using a combination of MPI with OpenMP in order to reduce the interprocess communication, and analyzed the scalability efficiency for field data case [8].

In order to bypass the complexity of the calculations [3] we used a simple algorithm [4] that exploits the huge calculation capacities offered by parallel systems. In the present paper the gravity anomalies are investigated because of the simplicity of the gravity problem [9] in geophysics. The focus of the study is the convergence and the scalability of the algorithm and how the solution is approximated during the iterations. The results are analyzed in terms of application runtime in parallel systems and in the character of the convergence process.

2. Methodology of the Work. The iterative inversion method analyzed in the paper is based on the algorithm CLEAN proposed by Högbom in 1974 [4]. In order to obtain solutions within a reasonable time the parallelization of the algorithm is used.

The initial results of the analysis of the algorithm using OpenMP are presented in [10]. In that paper the scalability of the parallelized algorithm is reported with the analysis of the number of iterations, the error and the runtime for different models running in serial and parallel modes in 16 nodes. In the present paper key results for the scalability of application up to 1024 parallel cores are reported, along with the in-depth analysis of the quality of the algorithm.

[*]This work makes use of results of the High-Performance Computing Infrastructure for South East Europe's Research Communities (HP-SEE), a project co-funded by the European Commission (under contract number 261499) through the Seventh Framework Programme. HP-SEE deals with multi-disciplinary international scientific communities (computational physics, computational chemistry, life sciences, etc.) stimulating the use regional HPC infrastructure and its services. Full information is available at http://www.hp-see.eu

[†]Polytechnic University of Tirana (nfrasheri@fti.edu.al).

[‡]Academy of Sciences of Albania (sbushati@yahoo.com).

Tests were undertaken with synthetic models, which permitted comparison of the approximation error in both the anomalous body and the anomaly itself. The model consisted of the 3D array of nodes representing the geosection situated under the 2D array of the ground surface points, where the field data were surveyed (in the Fig. 2.1 the 3D geosection is shown represented by the 3D array of nodes, with the anomalous prismatic body situated at its center under the 2D array of ground points where the anomaly of gravity is surveyed).

The geosection was divided in cuboid elements, each of them represented by the respective 3D node (the center of the element). In each iteration the core of the algorithm selected the node which gravitational effect (elementary anomaly) offered the best least squares approximation of the gravity anomaly at the surface, for a calculated mass density of the respective cuboid element. The mass density of the selected node was updated with a predefined quantity and its effect was subtracted from the anomaly. In order to parallelize the algorithm, the geosection was divided in sub-sections, each of them processed in one core. Best selected nodes from each sub-section were compared with each other to find the best node for the whole geosection.

The iterative process was designed as follows:

1. Start with a 3D geosection array of nodes with mass density initialized by zeros, and a 2D gravity anomaly array surveyed in the field
2. Fork parallel threads for each sub-section
3. Start parallel threads
4. Search the best node in each sub-section 3D array
5. Stop parallel threads
6. Compare selected nodes for each sub-section
7. Increment the density of the best selected node by a fixed amount (density step)
8. Subtract the effect of the modification of the geosection from the surface gravity anomaly;
9. repeat the steps (2):(3):(4):(5):(6):(7):(8) until termination conditions fulfilled.

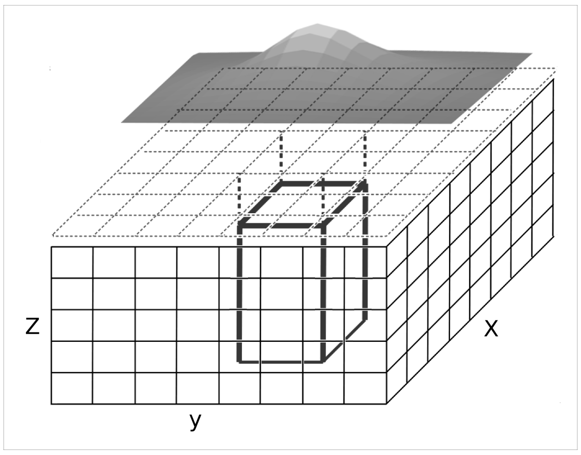


Fig. 2.1: The geosection model (thin line - 3D geosection array, dotted line - 2D surface profiles array, thick line - anomalous body).

The calculation of least squares in the step (4) was reduced in a 2D sub-array window of points over the node in question. The size of the window was correlated with the shape of the element anomaly depending on the element depth in order to consider only the area where the elementary anomaly had significant values greater than the floor $1/K$, where K is a predefined number. A sample of normalized anomalies (anomalous values divided by their maximum for each anomaly) for elements in different depths is presented in Fig. 2.2.

During each iteration the density of a single cuboid element of the geosection is updated with a predefined density step. Two alternatives for the iteration termination criteria were investigated:

1. when the decrease in the global least squares error becomes less than a very small predefined value;
2. when the elementary density giving the best approximation within the window results in less than half of the predefined density step.

The calculations for the effect of a single 3D node were considered as an elementary calculation block. The number of elementary calculation blocks for one iteration is equal to $N_{calc} = (N_x N_y N_z) \times (N_a N_b)$, where N_x, N_y, N_z are the number of nodes in linear edges of the 3D prismatic geosection array, and N_a, N_b are the

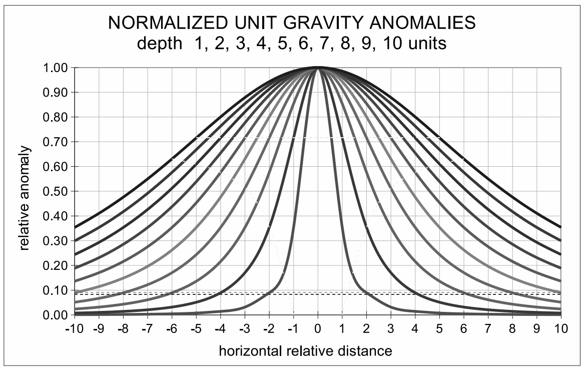


Fig. 2.2: Normalized elementary anomalies for unitary bodies in different depths 1:10 units; two horizontal lines at the level 0.1 define the least squares windows for floors 1/10 and 1/11.

number of points in linear edges of the 2D rectangular anomaly array. N was considered as a representative of the number of linear nodes / points, and the volume of calculations in one iteration resulted $O(N^5)$. The update of the best 3D node in each iteration was done with a fixed density step, and each node was updated until a limit mass density value was obtained. The quotient N_d of the mass density limit divided by the density step is one factor that defines the number of iterations. The overall order of elementary calculations blocks for the whole iterative process would be $O(N^5 N_d)$, the same stands for the runtime as well. Such high order of the volume of calculations made the parallelization obligatory in order to obtain inversion results for models of relatively high resolution.

The walltime T_w (difference between the time-stop and time-start of the program) was obtained using the OpenMP routine omp_get_wtime(). The linux *time* command was used to run the program in order to get the accumulative percentage of CPU usage. The runtime T_r (walltime in case of 100% CPU) was calculated as $T_r = T_w \times CPU\%/cores$.

Basic experiments were carried out for a geosection of $4000m \times 4000m \times 2000m$ digitized with 3D node arrays of step 400m, 200m, 100m and 50m. The anomaly under consideration was calculated in a similar 2D array of points for a vertical prismatic body with density 5 g/cm^3 situated at the center of the geosection (Fig. 2.1). More experiments with anomalies calculated for a geosection composed of two vertical prismatic bodies were carried out, as well as with real data from field surveys. Results included inversion geosections with distribution of densities and related anomalies, and the "carrot" presentation in 3D of distribution of densities in the central vertical section of the geosection for each iteration, showing the progress of the iterative process.

3. Evaluation of Iterative Inversion Process. Two alternatives mentioned before for the termination criteria of the iterative process were compared for arrays with step 200m and 400m, for density steps 0.1 g/cm^3, 0.5 g/cm^3 and 1.0 g/cm^3. Both alternatives gave similar number of iterations and of runtime, as shown in Fig. 3.1 for the array with step 200m.

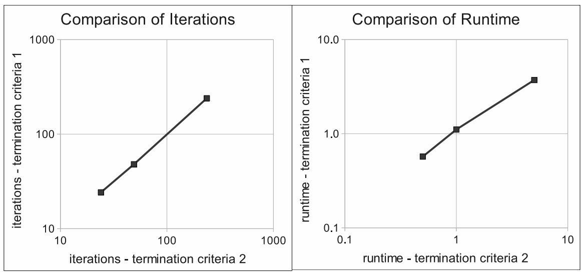


Fig. 3.1: Comparison of number of iterations (left) and runtime in seconds (right) for two termination criteria.

The global relative error achieved by each of termination criteria resulted dependent upon both the array step and the density step, as shown in Fig. 3.2.

Experiments with the single body geosection showed that, for window floors greater than 1/11, the window

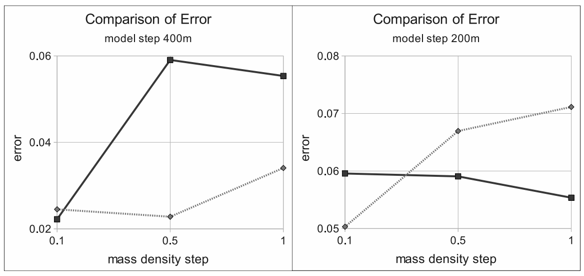


Fig. 3.2: Relative error by density step for array of 400m (left) and of 200m (right); lines in black represent the error obtained with the first criterion, in grey with the second criterion.

span of the anomaly of the surface element (depth 1 unit) was reduced to three nodes (Fig. 2.2) and the iterative process diverged (Fig. 3.3).

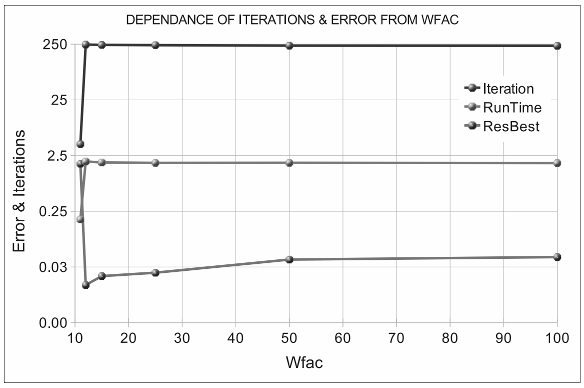


Fig. 3.3: Number of iterations, runtime and error for different windows floor factor.

Further tests showed that the algorithm had a tendency to form bodies with the largest mass density. Theoretically, a variation by a factor of k in the mass density of the body would be compensated for by a reversed variation in linear size of the body by a factor of $k^{(1/3)}$. In our model we used a body with a density of 5 g/cm^3 and tested the algorithm for maximal accepted densities varying from 1 - 9 g/cm^3. The variation in the size of the body at the central vertical 2D section is presented in Fig. 3.4.

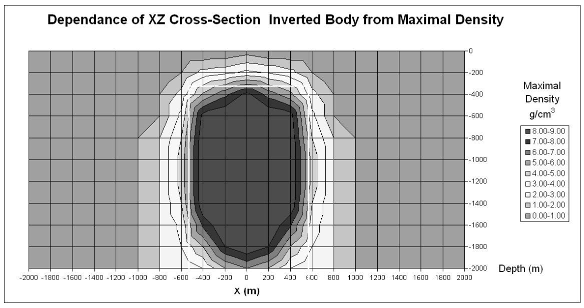


Fig. 3.4: Variation of anomalous bodies obtained by inversion for different maximal accepted mass density; the white rectangle represents the original body layout.

Both the theory and the findings supported the idea that keeping a maximal accepted mass density in the range of 2 - 3 g/cm^3 would be optimal, while higher values would simply lead to a reduction in the size of the body of at most 20%. At the same time the tendency to give larger mass densities is an indication how the algorithm optimizes the solution locally.

To understand better optimization process during the iterations, we combined the central vertical 2D geosection for each iteration in a single 3D image (Fig. 3.5).

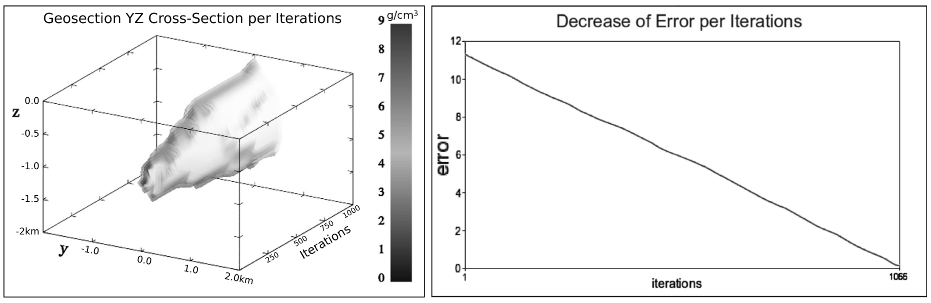


Fig. 3.5: Development of the anomalous body central section during inversion iterations (left) and the decrease in the anomaly approximation error (right).

The decrease per iteration in the error of approximation of the anomaly was linear, curving to constant in the final iterations (with a total of 1066 iterations), while delineation of the main section of the body was approximated with half of the iterations.

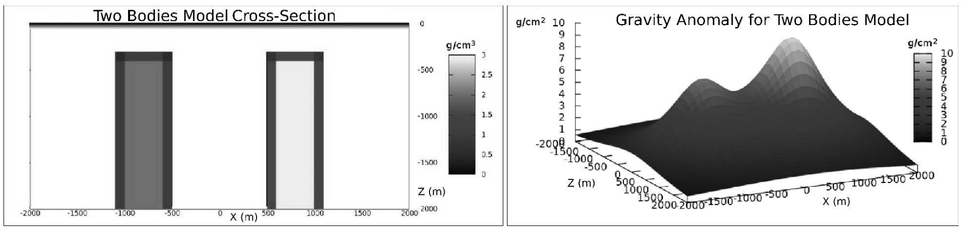


Fig. 3.6: Two bodies model geosection X-Z (left) and related the anomaly (right).

The case of inversion of an anomaly created by a geosection composed of two bodies presented an ill-posed problem. We used a geosection with two vertical prismatic bodies creating a bimodal anomaly (Fig. 3.6). The result of inversion, with a relative least square error of 10%, is shown in Fig. 3.7.

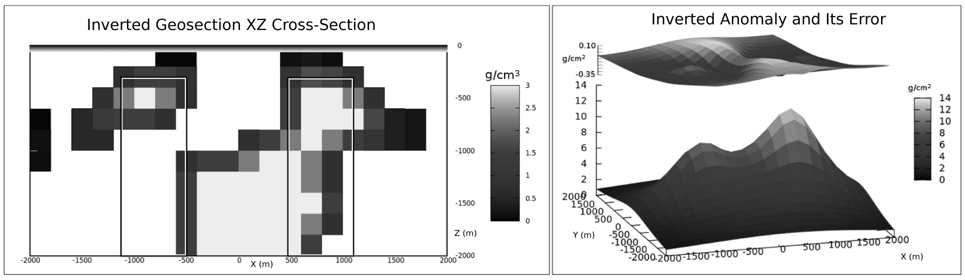


Fig. 3.7: Two bodies inverted geosection X-Z (left) and the related anomaly (right: bottom the anomaly, top: the error).

The error varied between -0.35 g/cm^2 and +0.1 g/cm^2 for the anomaly, which varied between 0.5 g/cm^2 and 10 g/cm^2. The inversion gave a three-body geosection, which intuitively may be deduced from the shape of the anomaly itself (apparently a regional anomaly combined with two local anomalies). The progress of delineation of bodies during iterations is given in Fig. 3.8.

The process begins with the approximation of the anomaly from a deep body (a) and only latter two shallow bodies are shaped (b,c). The algorithm tended to give point-like bodies, requiring careful interpretation in the case of anomalies from extended spatial structures (see e.g. Fig. 3.9, where the anomaly is created by extended magmatic structures). The anomaly is created by a combined effect of two mases, one (sedimentary, in cyan) with density less than the average and the other (magmatic, in red) with density greater than the average.

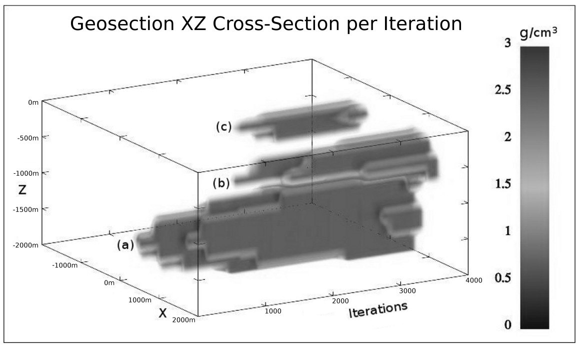


Fig. 3.8: Development of the anomalous central section during inversion iterations.

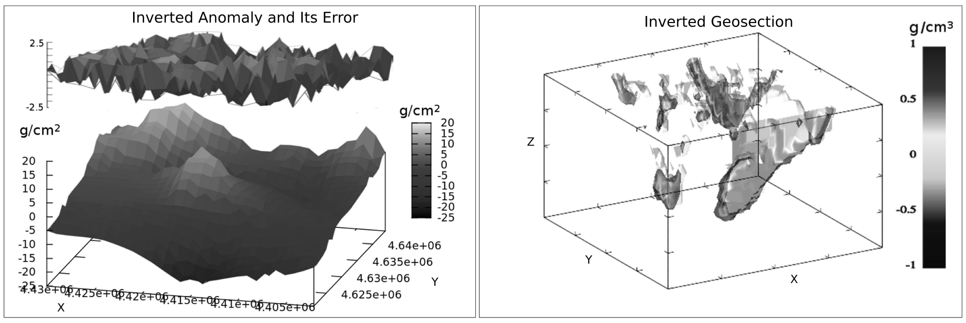


Fig. 3.9: Real case of field anomaly (left bottom), the error (left top) and the inverted geosection (right).

4. Convergence of the Iterative Inversion Process. Parallelization was undertaken using OpenMP, splitting in chunks the 3D array to search in parallel for the best node in each iteration. Parallel calculations were tested in two systems:

1. the HPCG centre at the Institute of Information and Communication Technologies (IICT-BAS) in Sofia, Bulgaria;
2. the NIIFI Supercomputing Centrr at University of Pécs, Hungary.

Tests in HPCG were done for the anomaly of a vertical prismatic body; with 1, 8 and 16 parallel cores; for geosection arrays with step 400m, 200m and 100m (corresponding to 11, 21 and 41 linear nodes) and density steps 0.1 g/cm^3, 0.5 g/cm^3 and 1.0 g/cm^3. Tests were done using the variation of the global error as termination criteria. The number of iterations as a function of the density of the 3D array is presented in Fig. 4.1. Increase in iterations slows down with the increase in the size of the problem. For comparison the line represents the order of O(N^4).

The relative least square error as a function of the density of the 3D array is presented in Fig. 4.2. Differently from the iterations, the mass density step had little impact on the error.

Further tests were done in the NIIFI centre using up to 1024 parallel cores. The number of cores permitted a model to be run with a 3D array step of 50m, which was not possible in HPCG. Given the fact that the resulting error was relatively independent of the mass density step, tests were done only for a density step of 1 g/cm^3. The number of iterations and the error for 3D array steps of 400m, 200m, 100m and 50m (respectively 11, 21, 41, and 81 linear nodes) are presented in Fig. 4.3.

The factor line represents the order of O(N^3). The results confirmed the slow down in the increase in the number of iterations when the spatial density of geosection nodes is increased. The error apparently does not depend strongly upon the density of the array.

The scalability was evaluated through comparing the runtime for different number of cores and model sizes, with a maximum of 1024 cores and a model resolution of 50m (Fig 4.4). Due to limitations in the availability of used HPC systems, it was not possible to test models with higher resolution.

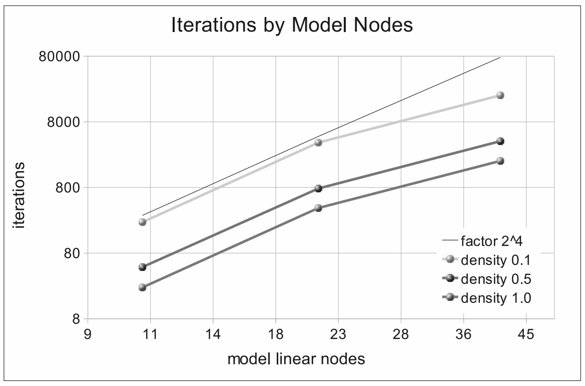


Fig. 4.1: Number of iterations per density of the 3D array; curves represent different mass density steps (the top-down order as in the legend).

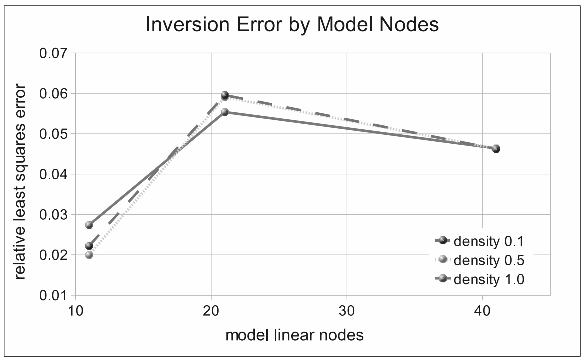


Fig. 4.2: Inversion error per density of nodes, curves represent different mass density steps (the top-down order as in the legend).

The maximal processor runtime that was achieved in parallel systems was 3.3 hours in the NIIFI and 21.6 hours in the HPCG.

5. Conclusions. Results of the calculations show that parallel systems may be used successfully for the inversion of geophysical anomalies, but at the same time the process of calculations remains tricky. Even simple algorithms as CLEAN when applied for 3D models required considerable HPC resources (number of cores and cpu runtime), while the results may be disputable in case of complex geosections, and the scalability of the algorithm would decrease when it runs in parallel systems shared by many users.

Utilization of OpenMP resulted in simplification of the programming, but the parallel systems resources available in southeastern Europe were unsuitable for this technology - only one site offered more that 16 parallel cores for OpenMP. Switching towards MPI - based solutions is obligatory and would permit running of software in HPC clusters and grid clusters that are more available than shared memory parallel systems.

Use of simple algorithms in parallel systems was, in terms of runtime and error obtained, successful for geosections with massive bodies; for geosections with thin structures detailed 3D arrays have to be used, which leads to an increase in the runtime to levels that may be difficult to be supported by existing parallel systems available in the region.

In our models the minimal spatial differentiation achieved was 50m using 256 and 512 parallel cores (available only in one site) for a runtime of order of hours. Tests with multiple bodies and real field data reconfirmed the need of using initial solutions defined on the basis of other geological factors, and the need for careful interpretations of results.

Acknowledgments. The authors thank the HPC-BG and NFII teams for all the help and support during exploitation of respective parallel systems.

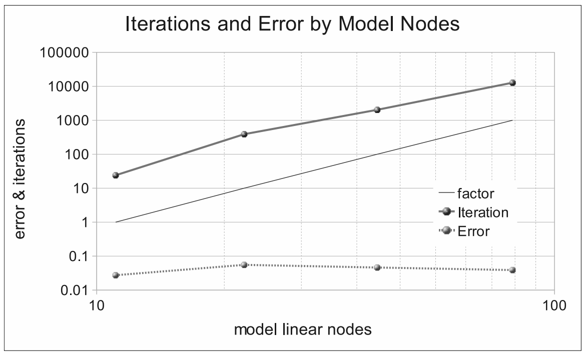


Fig. 4.3: Number of iterations and inversion errors per density of 3D array (the top-down order of curves as in the legend).

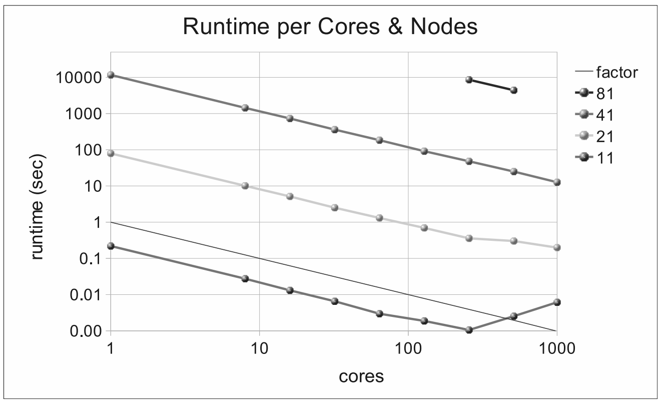


Fig. 4.4: Scalability of the algorithm and the runtime for number of cores and size of model (the top-down order of curves as in the legend).

REFERENCES

[1] J. Hadamard, *Sur les prolemes aux derivees partielles et leur signification physique*, Bull Princeton Univ., 13, l-20, 1902.

[2] P. Shamsipour, M. Chouteau, D. Marcotte, P. Keating, *3D stochastic inversion of borehole and surface gravity data using Geostatistics*, in EGM 2010 International Workshop, Adding new value to Electromagnetic, Gravity and Magnetic Methods for Exploration Capri, Italy, April 11-14, 2010.

[3] F.J. Wellmann, F. G. Horowitz, E. Schill, K. Regenauer-Lieb, *Towards incorporating uncertainty of structural data in 3D geological inversion*, in Elsevier Tectonophysics TECTO-124902, 2010. http://www.elsevier.com/locate/tecto (retrieved 07 Sept 2010).

[4] J. A. Högbom, *Aperture Synthesis with a Non-Regular Distribution of Interferometer Baselines*, in Astr. Astrophys. Suppl., 15, 417, 1974.

[5] P. Rickwood and M. Sambridge, *Efficient parallel inversion using the Neighborhood Algorithm*, in Geochemistry Geophysics Geosystems Electronic Journal of the Earth Sciences. Volume 7, Number 11, 1 November 2006.

[6] M.H. Loke and P. Wilkinson, *Rapid Parallel Computation of Optimized Arrays for Electrical Imaging Surveys*, in Near Surface 2009 15th European Meeting of Environmental and Engineering Geophysics Dublin, Ireland, 7-9 September 2009.

[7] Hu Zuzhi, He Zhanxiang, WangYongtao, Sun Weibin, *Constrained inversion of magnetotelluric data using parallel simulated annealing algorithm and its application*, in SEG Expanded Abstracts / Volume 29 / EM P4 Modeling and Inversion, SEG Denver 2010 Annual Meeting.

[8] G. Wilson, M. uma, and M. S. Zhdanov, *Massively parallel 3D inversion of gravity and gravity gradiometry data*, in PREVIEW - The Magazine of the Australian Society of Exploration Geophysicists, June 2011.

[9] W. Lowrie, *Fundamentals of Geophysics*, Cambridge University Press 2007.

[10] N. Frasheri and B. Cico, *Analysis of the Convergence of iterative Gravity Inversion in Parallel Systems*, in ICT Innovations 2011 Conference, Macedonian Academy of Sciences and Arts (MANU), Skopje, 14-16 September 2011.

Edited by: Marcin Paprzycki
Received: May 1, 2012
Accepted: June 15, 2012

Scalable Computing: Practice and Experience
Volume 13, Number 2, pp. 159–171. http://www.scpe.org

ISSN 1895-1767
© 2012 SCPE

NEW SPARSE MATRIX STORAGE FORMAT TO IMPROVE THE PERFORMANCE OF TOTAL SPMV TIME

NEELIMA B.*, PRAKASH S. R.* AND G. RAM MOHANA REDDY*

Abstract. Graphics Processing Units (GPUs) are massive data parallel processors. High performance comes only at the cost of identifying data parallelism in the applications while using data parallel processors like GPU. This is an easy effort for applications that have regular memory access and high computation intensity. GPUs are equally attractive for sparse matrix vector multiplications (SPMV for short) that have irregular memory access. SPMV is an important computation in most of the scientific and engineering applications and scaling the performance, bandwidth utilization and compute intensity (ratio of computation to the data access) of SPMV computation is a priority in both academia and industry. There are various data structures and access patterns proposed for sparse matrix representation on GPUs and optimizations and improvements on these data structures is a continuous effort. This paper proposes a new format for the sparse matrix representation that reduces the data organization time and the memory transfer time from CPU to GPU for the memory bound SPMV computation. The BLSI (Bit Level Single Indexing) sparse matrix representation is up to 204% faster than COO (Co-ordinate), 104% faster than CSR (Compressed Sparse Row) and 217% faster than HYB (Hybrid) formats in memory transfer time from CPU to GPU. The proposed sparse matrix format is implemented in CUDA-C on CUDA (Compute Unified Device Architecture) supported NVIDIA graphics cards.

Key words: Graphics Processing Unit (GPU), data parallelism, sparse matrix, SPMV computation, compute intensity, memory transfer time, CUDA-C, NVIDIA Graphics Card.

1. Introduction.

Graphics processors (GPUs) are proved to be good for data parallel applications. GPUs have also been proved as a good choice for irregular memory access applications that have high data parallelism. Example of such applications include sparse matrix vector multiplication, graph algorithms etc. Several scientific computations use SPMV computation as a main kernel. Improving and optimizing SPMV computation is still a research focus for the new hardware architectures. The sparse storage format used in SPMV determines the performance of the application. The steps involved in SPMV computation on GPU are: data organization (to make memory access efficient on GPU), memory transfer of input data from CPU to GPU and SPMV computation on GPU. The result is very small in size, that need to be sent back to CPU from GPU and this small value is not considered in this work. The sparse storage formats used for CPUs cannot deliver good performance when used for SPMV computation on GPU. So, many GPU specific new formats and optimizations are evolving. Most of the formats and optimization methods have taken only SPMV computation time on GPU into consideration and tried to optimize GPU performance. As SPMV computation is performed on GPU, there are common overheads in terms of data organization time by GPU or CPU, data transfer from CPU to GPU. At the same time, the computation power available on GPU is not negligible and should be utilized for the high performance applications. This paper proposes a new sparse storage format that can reduce the time of data organization and memory transfer, reducing the overall computation time of SPMV. The new format is called as Bit Level Single Indexing (BLSI). BLSI implementation is done on CUDA and proved good for GPU architecture. This format reduces the number of bytes required per flop, reducing the compute intensity or ratio of bytes to flops. It also saves on cache and/or register usage per thread on GPU. The total time comparison shows that BLSI is 2x to 112.6x faster than HYB (HYBrid format) when total time is considered as the SPMV time. The BLSI sparse matrix representation is upto 204% faster than COO, 104% faster than CSR and 217% faster than HYB formats in memory transfer time from CPU to GPU

The paper is organized as follows. An overview of the GPU architecture and sparse formats are given in Sect. 1 and 2 respectively. Section 4 highlights the importance of SPMV optimization on GPU by giving the related work. Section 5 details the new format generation and uses. Section 6 gives the experimental set up. Section 7 gives results and analysis and concludes with future work in Sect. 8.

2. GPU Architecture.

Usage of GPUs for general purpose computations have accelerated when NVIDIA introduced CUDA, a general purpose parallel computing architecture. A CUDA device or the GPU is connected to CPU through host interface. CUDA device consists of a set of Streaming Multiprocessors (SMs), each consists of an instruction unit and a shared memory along with a set of Streaming Processors (SPs). Each core can preserve number of thread contexts, specific to the architecture. CUDA has zero-overhead scheduling, that is Fmaintained by tolerating the data fetch latency by switching between threads [1].

*Department of Information Technology, National Institute of Technology, Karnataka, India. (reddy_neelima@yahoo.com) Questions, comments, or corrections to this document may be directed to this email address.

$$\underset{A}{\begin{pmatrix} a & 0 & b & 0 \\ 0 & 0 & c & 0 \\ 0 & d & 0 & 0 \\ e & 0 & 0 & f \end{pmatrix}}$$

Fig. 3.1: Example sparse matrix A

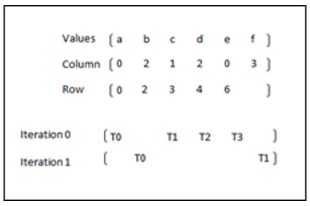


Fig. 3.2: CSR representation sparse matrix and its thread access pattern on GPU

Parallel region of CUDA program is called as kernel. The CUDA API (Application Program Interface) is used to create parallel threads to be executed on the hardware. The kernel is partitioned into a grid of thread blocks that can execute in parallel. The programmer can define the dimensions of the grid and block. The SPs are grouped in SIMD (Single Instruction Multiple Data) fashion and CUDA follows SIMD. Thread blocks are distributed evenly on the multiprocessors. Threads are logically grouped into warps. A warp consists of 32 threads that can execute a single instruction. Each SM executes one warp at a time. Different warps within SM are time shared on the hardware resources. Thread divergence is created with the usage of conditional instructions that serializes the threads.

CUDA device has hierarchy of memories. The device memory is called as global memory. Memory request of a half warp (16 threads) are served together, this is called as coalescing. The request from all the threads of a warp is coalesced into one memory transaction if they are accessing the addresses in the same segment. Once the addresses are accessed by the half warp in one segment, it is called as fully coalesced. This is one of the optimization that is looked into for any CUDA computations. The shared memory is accessible by threads of the same block. Along with this, set of registers are shared by the threads of a block. The constant and texture memories are read only memories in global space with on-chip caches. The programmer can bind these regions to read only data before launching a kernel [2].

3. Sparse Matrix Formats on GPU. The SPMV computation involves a sparse matrix A multiplied by a dense vector x, represented as $y{=}Ax$. The standard formats like CO-Ordinate (COO), Compressed Sparse Row (CSR), ELL (ELLPACK) and HYB (HYBrid) [10] formats are considered in this paper to compare with the proposed format and explained briefly here. The other formats are built on these standard formats that are given under related work section. Bell and Garland [10] gave a detailed study of sparse formats and their access pattern on GPU.

3.1. Compressed Sparse Row (CSR) Format. A sample sparse matrix A is shown in Figure 3.1 and Figure 3.2 shows the CSR representation and its thread assignment on GPU hardware. Here, one thread per row is launched. In Figure 3.2, T0 through T3 reads the elements in each row, first iteration will give full throughput, but in the next iteration only two threads read the elements. The column indices are not accessed simultaneously even though they are stored contiguously, which causes poor performance of this format.

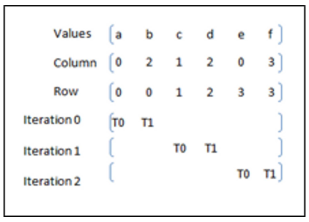


Fig. 3.3: COO representation of sparse matrix and its thread access pattern on GPU

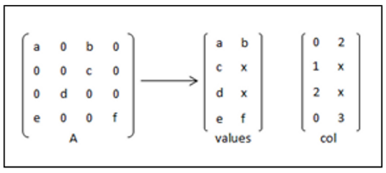


Fig. 3.4: ELL representation of sparse matrix

3.2. CO-Ordinate (COO) Format. In COO format, there are three one-dimensional arrays of same size as that of the number of non-zeros in the matrix. It causes an overhead in terms of memory transfer from CPU to GPU. The COO representation and thread access pattern is shown in Figure 3.3. It can be noted that warp sized (here warp size is 2) number of threads work on the non-zeros in each iteration. Reduction or atomic operations can be used across threads. Reduction operation is used to compute the sum of an array of numbers in parallel. Atomic operations allow multiple threads to perform concurrent read-modify-write operations in memory without conflicts. The syntax of atomic operation in CUDA is as follows: float atomicAdd(float* address, float val).

3.3. ELL Format. ELL representation is shown in Figure 3.4 and thread assignment is given in Figure 3.5. This technique is suitable for vector architectures. Column major access is preferred as it offers better coalescing, and shared memory can be used with ease since there wont be any bank conflicts. As shown in Figure 3.5, the threads are launched in column major order. After each iteration, the threads advance to the next column for execution.

3.4. Hybrid (HYB) Format. Hybrid format is combination of sparse matrix formats proposed by [10]. HYB uses combination of ELL and COO. The HYB structure is shown in Figure 3.6. ELL and COO is faster for SPMV in many cases, but it has a CPU-GPU memory transfer overhead when compared with other formats, since it requires five memory transfers; two for ELL and three for COO. The Thread access pattern is a combination of the access pattern of ELL and COO.

ELL and COO combination as used in HYB format is preferred because ELL is proved to be good when the difference in number of non-zeros in each row is negligible, and COO is proved to give a modest performance when the number of non-zeros per row is variable. It can be seen that the portion of the row till size L (calculated empirically) is considered to be the ELL portion of the row and if the size of the row exceeds L, it is considered as the COO portion of the row and is stored in ELL format. Alternatively, the format can take entire row as ELL if its size is less than or equal to L or as COO if its size is more than L.

4. Related Work. SPMV performance improvements and optimization based publications are on rise in recent times. The importance of communication overhead in high performance application and the need of optimizing this overhead were studied extensively in the literature as follows. Ravi et. al [3] have proposed a heterogeneous BLAS library for SPMV computation considering communication bandwidth as one of the parameters to tune the applications parameters according to the architecture in a heterogeneous system. Our work optimizes this bandwidth limitation by using a new data structure for the sparse matrix representation for a single GPU. Xingfu wu et. al [4] proposed hybrid optimization methods for scientific and compute intensive applications for CMP clusters. They map processors per node optimally and similarly this work also maps computation to the threads to increase the performance of the applications. Vuduc et.al [5] discussed three main applications for which GPU has some limitations. One of the applications discussed is SPMV operation which

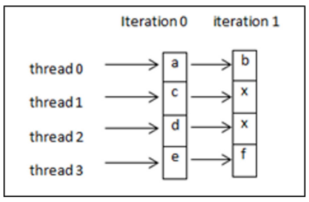


Fig. 3.5: ELL thread access pattern on GPU

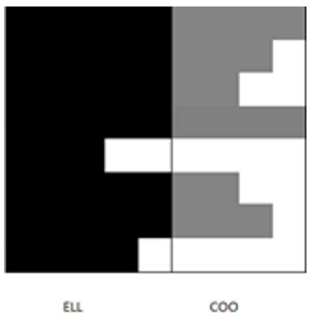


Fig. 3.6: HYB structure of sparse matrix

is bandwidth limited and mentioned that porting this application is beneficial if CPU to GPU communication is reduced or removed by some methods. The method proposed in this paper considers the memory transfer time and improves it.

Lee et.al [6] have analysed main kernels that appear in most of the applications and mentioned the requirement of optimizations on and off the device. SPMV is one kernel that can improve performance if bandwidth limitations are overcome. BLSI method optimizes data organization time and memory transfer time that can strengthen the usage and need of GPU for general purpose computation. Gregg and Kim [7] proposed that moving computation to data improves the application performance than always moving data so that data movement overheads are reduced. They proposed to use system intelligence and develop an automated tool to control the assignment of the computation. BLSI method also optimizes data movement by reducing the amount of data to be given to the computation with easy and flexible implementation. Michael et. al [8] have also considered bandwidth bound applications for GPU and proposed an API for creating DMA warps that exclusively handles memory transfers from off chip memory to on chip memory. Our method of optimizing memory bandwidth is external to the device. Thomas et.al [9] proposed an automatic tool that consists of runtime library and compiler optimizations to optimize the number of communications from CPU to GPU. The tool proposed by them combines all the memory copies to the GPU of multiple kernels. This method is beneficial if the kernels use few common data. Our method is to optimize communication especially for SPMV, which is very commonly used in most of the scientific and engineering applications.

Storage format of sparse matrix is very important in determining the performance of SPMV. Various proposals of sparse matrix storage formats are discussed here. Bell and Garland [10] proposed a new GPU suitable sparse matrix storage format namely HYB which is a combination of ELL and COO. The kernel time of HYB is improved a lot at the cost of high data organization and memory transfer time. The data structure proposed in this paper gives improvements in overall time of SPMV computation, of course at the cost of SPMV kernel time. Blelloch et al. [11] studied SPMV on vector machines. Choi et al. [12] proposed the BELLPACK representation that suits for matrices with dense blocks. Yang et al. [13] proposed optimizations using texture to increase data locality that improves SPMV performance. Monakov et al. [14] [15] implemented blocked SPMV and Sliced ELLPACK in which a slice of the matrix, a set of adjacent rows, are stored in ELL format.

Vazquez et al. [16] proposed ELLPACK-R format that uses ELL format with an array containing the length of each row. They assigned multiple threads to a row to balance the computations of threads. Anirudh et al. [17] and Kiran and Kishore [18] considered combination of CSR and ELL formats for storing the matrix. Dziekonski et al. [19] have proposed sliced ELLR-T data structure. Most of the formats proposed increases the performance on chip and not considered other overheads like memory transfer and data organization time. The communication optimized sparse data structure by name CSPR was proposed by Neelima and Prakash [20]. The method was not scalable and not tested with CUSP results.

Various optimizations to the existing formats can further improve the performance of SPMV computation on GPU. The work by Yelicki et al. [21] and Vuduc et al. [22, 23, 24, 25] gave extensive optimizations and auto-tuning for sequential machines. Low-level code optimizations and data structure optimizations for single core and parallelized optimizations for multi-core architectures is given by Williams et al. [26], [27]. Brahme et al. [28] proposed a greedy extraction of dense sub matrices that load balances and overlaps communication and computation, so, reduces memory traffic and hides communication latencies. It is an on chip optimization, uses greedy algorithm and has less scope for scalability. Zein and Rendell [29] proposed an analysis and selection tool that selects the best performing implementation for SPMV. Wang et al. [30] proposed optimizations for SPMV computation on CUDA by giving optimized CSR storage, thread mapping etc. Baskaran and Bordawekar [31] also proposed optimizations to improve SPMV performance by using synchronization free parallelism, optimized threads mapping, data fetch and data reuse. This work proposed a new storage format for the sparse matrix that improves the performance of the application by optimizing the data organization and memory transfer time from CPU to GPU. This work also optimizes the computation by optimizing the thread mapping.

5. New Format for Sparse Matrix Representation.

The concept of single index representation is implemented at the bit level and hence the name Bit-Level Single Indexing (BLSI) is given. This is also implemented at the integer level by using division and modulus operations, but for the ease of programming and optimal index generation, bit level implementation is chosen for observing the results. The results or the total time is same in both bit level and integer level implementations.

5.1. Index Generation.

Contrast to many standard sparse matrix formats like COO, CSR etc., which use one array or one data structure for column index and another to keep information about the row, BLSI method uses only single array or data structure to store the indices by embedding the column information in the bits of row indices information. Hence, this method needs only one array of size equal to the number of non-zero elements to represent the indices. If the column index is also big and could not fit into the remaining bits that are available, then offset is used to keep track of column index while using small value to represent the column index that fits into the array. Offset size will be much smaller compared to the size of the second array in COO format (ITER<<<nnz) and smaller than CSR format pointer array (ITER<<ptr) to mention a few. The size of the offset array, ITER, is computed as:

$$ITER = (numEle + (THPB * BLOCKS) - 1)/(THPB * BLOCKS) \tag{5.1}$$

Terminology used:

- ITER : Number of iterations required and size of offset array
- THPB : Threads per block
- BLOCKS : Number of blocks launched in the grid
- numEle : Number of non-zero-elements
- BIT_SHIFT : The power of two taken (in the example given, it is 17)
- Offset[] : Array to store the offsets for different ITER
- newRow[] : Array that contains the value of row index of corresponding non-zero element
- newCol[] : Array that contains the value of column index of the corresponding non zero element
- index[] : Array that contains the row index and column index embedded into few bits
- value[] : Array that contains the non-zero-value
- REM_AND : (2 power of BIT_SHIFT)-1
- B[] : Dense vector array
- dotd : Contains result of multiplication of non-zero-value with the corresponding element in the vector

5.2. BLSI: Bit Level Single Indexing.

BLSI is a new format proposed to represent sparse matrix indices to reduce memory transfer overhead in the memory bound SPMV computations. The main operations involved in SPMV computation on a GPU are organizing the data (for enabling global coalescing by changing the data layout and other optimizations to get performance benefit from GPU), sending the data from CPU to GPU and executing SPMV computation on GPU. The memory transfer time for the results from GPU to CPU is very minimal and not considered here. The SPMV computation time is given as total time of all these operations and through this new data structure improvements are seen in all these steps of SPMV computation.

Through this new method, this work presents optimizations at three levels as follows:

- At the data format or data mapping level
- At the communication level between CPU and GPU
- At the computation level while assigning threads for computation

Algorithm 5.2.1 Index generation in BLSI format

```
for(j=0;j<ITER;j++)
{
offset[j]=newCol[j*THPB*BLOCKS];
for(i=j*THPB*BLOCKS;i<((j+1)*THPB*BLOCKS) && (i<numEle);i++)
{
index[i]=((newCol[i]-offset[j])<< BIT_SHIFT)+newRow[i];
}
}
```

The number of bits required for the single index representation is the sum of BIT_SHIFT size and the offset [] data size. The size of BIT_SHIFT is the immediate next power of 2 of the row size. For example, if the number of rows are 86, 000, then 2^{16} = 65,536 and 2^{17} = 131,072. So BLSI considers BIT_SHIFT size as 17. As mentioned earlier, BLSI needs offset array to reduce the number of bits to represent row and column information into a single value. The column index is left shifted by BIT_SHIFT size and then added with row index. The index generation algorithm is given in Algorithm 1. The column index added with row index and the row index is extracted on GPU in SPMV GPU kernel as given in Algorithm 2.

Algorithm 5.2.2 SPMV computation using BLSI format

```
unsigned int i =blockDim.x * blockIdx.x + threadIdx.x;
for( ;i<N; i+=BLOCKS * THPB )
{
row = index[i] & REM_AND;
col = (( index[i] >> BIT_SHIFT) + offset[i / (THPB*BLOCKS)]);
dotd = value[i] * B [col];
atomicAdd( result+row, dotd); }
```

The BLSI index can also be computed as row-index * n + col-index, which is equivalent to the index calculated in Algorithm 1. The index of row and column computation from single index as shown in Algorithm 2 is equivalent to obtaining row by doing division and modulus operation on BLSI index with size of the matrix to get row and column index respectively as shown in Figure 5.1.

5.2.1. Data Format Optimization. The proposed BLSI format optimizes the data organization time and also the data storage requirement for the memory bound SPMV computation. The input data to SPMV kernel on GPU needs reorganizing the data to enable the global coalescing and other optimizations to get the actual performance improvements on the GPU. BLSI method requires pre-processing to restructure the matrix into a single index based matrix from .mtx, Matrix Market Format, a standard file extension used in many benchmark matrix data that uses COO format to represent the values in the matrix. But this pre-processing is involved with any other sparse data representation other than COO, which needs to be generated from COO. For example ELL, CSR etc. formats have to be generated form the Matrix Market Format i.e, COO. The overhead involved in generating BLSI format is still very less compared to the other formats. Table 1 shows the total time for BLSI and HYB format. Total time includes data reorganization time, Sending data from CPU to GPU and the kernel computation on GPU. HYB format is highly optimized format on the GPU so far. The

val	= [a b c d e f]
BLSI-Index	= [0 2 6 9 12 15]
BLSI-Index	= row-index * n + column-index
row-index	= BLSI-Index / n
column-index	= BLSI-Index % n

Fig. 5.1: Sample data format representation of sparse matrix A in BLSI method, single index calculation and extraction of the row and column index calculations represented at the data type level. n is the size of the square matrix.

Table 5.1: Total and kernel time executions for BLSI and HYB formats

Input Matrix	*BLSI − total* time in ms	*HYBtotal* time in ms	*Ratio :* HYB/BLSI	*HYB − Kernel* time in ms	*BLSI − Kernel* time in ms
web	21.307	581.689	27.301	0.847	4.637
ship	21.399	2191.501	102.411	1.426	3.396
scircuit	7.478	248.329	33.208	0.459	1.371
rma	10.098	741.014	73.379	0.617	2.207
rail	55.504	108.001	1.946	3.772	20.656
qcd	8.686	546.219	62.888	.227	2.287
pwtk	31.701	3344.274	105.493	1.502	3.607
pdb	12.674	1427.531	112.633	0.971	1.479
mc2depi	14.084	604.106	42.894	0.246	2.472
mac	11.134	420.245	37.744	0.639	2.178
dense	13.949	29.763	2.134	0.849	1.994
cop20k	9.851	796.334	80.838	1.047	2.412
consph	17.356	1951.671	112.452	0.889	2.427
cant	12.384	1361.669	109.959	0.765	1.693

total time comparison shows that BLSI is 2x to 112.6x faster than HYB when total time is considered as the SPMV time. If we compare the total time (that includes the kernel time) with the kernel time of the respective sparse formats then total time is almost 2.6x to 8x times the kernel time on GPU for BLSI and 28x to 2452x times the kernel time for HYB format. The total time is observed by using the CUDA event recoder for the CUSP library operations. The improvements in communication between CPU and GPU are shown in Sect. 7. BLSI kind of new formats are desired for the GPUs that are massively data parallel architectures to show the overall benefit of using the GPU for data parallel computations. The problem of CPU-GPU communication scales up as the number of GPUs used increases in a system. If the matrix is very large and does not fit into the chip storage, then SPMV performance still degrades and BLSI format gives better overall timing in this case too. Hence the proposed new data representation is an optimized data format for the GPUs.

5.2.2. Communication Optimization. The Matrix Market format is the most used standard format that uses COO data format. An optimized library like CUSP [32] etc., uses .mtx files and builds other formats from COO. So, the communication time between CPU and GPU is the time taken to transfer two index arrays for any format. The BLSI sparse matrix representation is up to 204% faster than COO, 104% faster than CSR and 217% faster than HYB formats, in memory transfer time from CPU to GPU. The results of comparison for the different matrices are given in Sect. 7.

5.2.3. Computation Optimization. BLSI uses atomicAdd() computation where all the threads need to synchronize to add the row-wise product to single sum. To optimize even this computation time using atomic operations, the thread assignment is modified as follows. In the pre-processing stage, BLSI does some changes to the COO matrix (from Matrix Market file) [33], to change the data access pattern that in turn optimizes the thread computations. The detailed explanation is as follows. For the matrix given in Figure 3.1, the thread assignment for an SPMV kernel is shown in Table 5.2.

If the data access pattern is not changed, two consecutive threads take the computation of values a and b of row 1. Assume that both finish the computation at the same time. These two values need to be added to a single value that represents the sum of the products of that row. Only one thread can access the sum

Table 5.2: Thread assignment: continuous threads are assigned for values with in a row

Old Thread Assignment						
Thread ID	1	2	3	4	5	6
Val	a	b	c	d	e	f
Row	1	1	2	3	4	4

Table 5.3: Data representation of .mtx file after changed by BLSI method

Row	**Col**	**Val**
1	1	a
4	1	e
3	2	d
1	3	b
2	3	c
4	4	f

as it is an atomic operation. The problem here is two threads finished the work at the same time but sum of products delays the overall result because of the atomic operation. To overcome this, BLSI uses a different data access pattern that delays the single row product computations so that the conflict to write to the row wise sum is delayed and in turn the row wise sum is available to all the products when they are completed with the computation. This technique has boosted the performance to much higher values. The time of execution is dominated by the row that has large non-zero-values.

To improve the performance using BLSI, BLSI changes the data representation of the .mtx file as follows. The Matrix market file is sorted based on column indices rather than row indices. Hence the .mtx representation will be changed as shown in Table 5.3 for the matrix given in Figure 3.1.

The thread assignment is done column wise here. Threads that belong to the same row need not wait for atomic operation, reducing computation time. The atomic operation of multiple threads is delayed that in turn improves overall performance of SPMV kernel. The thread assignment is shown in Table 5.4. By changing the access pattern, it gives up to 92% improvement in SPMV kernel computation than the previous access pattern.

These optimizations used are external to the device. As shown in Sect. 7, the results are promising in-terms of memory transfer time form CPU to GPU, overall GPU computation time that includes memory transfer and kernel executions time, new format generation time and also overall program execution time. Hence BLSI can be used as one of the sparse matrix formats that better utilizes the device.

6. Experimental Setup. This section describes the experimental setup used. The workload selected and workload parameters used in the experiments are given. Monitors used for the observation of outputs are listed. The specifications of hardware and software used for the experimentation are given in Table 6.1.

6.1. Workload. The input matrices used are the same workbench used by William et al. [26], [27]. These are the real data observed form the experiments and posted in University of Florida Sparse Matrix Collection [33]. The input workload use and its characteristics are given in Table 6.2.

6.2. Workload Parameters. The input matrices are evaluated with the proposed methods. The workload is characterized by performance observed in GFlops (10^9 Flops) and bandwidth in GBytes (10^9 Bytes). The time is measured for the kernel execution and these values are derived with the details of the matrices used.

GFlops is computed as follows:
GFlops = ((2* nnz)/(kernel execution time in milliseconds * 1000000))

Bandwidth is computed as follows:
Bandwidth = ((3*nnz) + (2*# of rows)*4)/(kernel execution time in milliseconds * 1000000)

To compare the communication time between CPU and GPU, only the time of communication or time for the memory transfer is taken into the consideration. The total time is observed as the time for the data organization of the input, memory transfer time form CPU to GPU and the SPMV kernel execution time on the GPU. For observations, CUDA event recorder is used for GPU related computations and CPU timer is used for the computations on CPU.

Table 5.4: Thread assignment: thread 1 access first value of row1, thread 2 access first value of row 2 and goes on

New Thread Assignment						
Thread ID	1	2	3	4	5	6
Val	a	e	d	b	c	f
Row	1	4	3	1	2	4

Table 6.1: Specifications of hardware and software used in experiments

System/Hardware Specifications	
Processor name and code name	Intel Core i7 2600, Sandy Bridge
Processor specification	Intel (R) Core(TM) i7-2600 CPU @ 3.40GHz
Graphics Interface	PCI-Express
Graphics processor name and code name	NVIDIA GeForce GTX 470-GF100
PCI-E link width	X16
Memory type	DDR3
Memory size	1280MB
Software Specifications	
Windows Version	Microsoft Windows 7(6.1) Service pack 1(Build 7601)
DirectX Version	11.0
Programming platform	Visual Studio 2010
CUDA SDK version	3.2 and 4.0

6.3. Monitors. NVIDIA provides ParallelNSight to profile the CUDA programs. The Communication time between CPU to GPU is taken from the profiler memory copy time from host to device. CUDA C also provides an event recorder to observe the elapsed time. For the GFlops and Bandwidth computation that are given as part of program, CUDA event record is being used to measure the kernel time. It is also been verified that the time taken by the CUDA event and the profiler for the kernel execution are same. Hence the mode of observation done is valid.

7. Analysis and Interpretation of Results. This section compares the proposed BLSI format against the most commonly used formats like COO, CSR and HYB. The results are given for the following comparisons.

Communication time (or) Memory transfer time: The memory copy (memCopy for short) time between CPU and GPU is compared for all the matrices given in the workload. They are compared by considering the time of memCopy in milliseconds. Figure 7.1, shows that BLSI takes less time for memCopy in all the cases. BLSI is up to 107% better than CSR, 204% better than COO and 217% better than HYB when compared for memCopy time of sparse matrix data from CPU to GPU. The percentage of variations in memory transfer time of COO, CSR and HYB with respect to BLSI is shown in Figure 7.2. In scircuit, the number of elements per row is very small and hence CSR ptr and BLSI offset sizes become same and also the kernel time of CSR is better for such matrices. In general, for a matrix with very few elements per row or few rows with large elements, BLSI method will not perform well.

Performance Observation in time of execution by considering the kernel + memCopy time of SPMV Computation: Figure 7.3 compares four formats with respect to SPMV computation that is, CPU to GPU communication time plus the kernel time. The total time taken by BLSI format in matrices scircuit, rail is more because, they have more nonzero elements distributed in very few rows. So the computation time taken by BLSI is higher, because BLSI uses atomicAdd() for row wise sum. The BLSI (our method) outperforms than all other methods for different structures of the matrices, when SPMV computation time and device memCopy time are considered. As explained earlier, this format was proposed to reduce the CPU-GPU communication, this optimization has resulted in overall better performance also. BLSI is 80% better than CSR, 164% better than COO and 161% better than HYB (as shown in Figure 7.3) when both the memory transfer time and kernel time are considered.

Figure 7.4 compares different sparse storage formats by considering total time as the data organization time, memory transfer time and the kernel time. The HYB results are not shown in Figure 7.4, because it deviates the graph. The values are given in Table 5.1. The total time in mac, scircuit, web and mc2depi of BLSI is little more than total time of CSR because offset array used in BLSI and kernel time dominates by the longest row reduction. As the number of elements per row is very small in these matrices, CSR kernel performance is better

Table 6.2: Specifications of hardware and software used in experiments

Matrix	Rows	NNZ	NNZ/Row	Description
cant	62,451	4,007,383	64	FEM cantilever
consph	83,334	6,010,480	72.1	FEM concentric spheres
cop20k_A	121,192	2,624,331	21.6	Accelerator cavity design
dense2	2000	4,000,000	2,000	dense matrix in sparse format
mac_econ_fwd500	206,500	2,100,225	3.9	Macroeconomic model
mc2depi	525,825	2,100,225	3.9	2D Markov model of epidemic
pdb1HYS	36,417	4,334,765	119.3	protein data bank 1HYS
pwtk	217,918	11,634,424	53.3	pressurized wind tunnel
qcd5_4	49,152	1,916,928	39	quark propagators (QCD/LGT)
rail4284	4,284	11,279,748	2,632.9	Railways set cover, constraint matrix
rma10	46,835	46,835	50.6	3D CFD of Charleston Harbor
scircuit	170,998	958,936	5.6	Motorola circuit simulation
shipsec1	140,874	7,813,404	55.4	FEM Ship section / detail
webbase-1M	1,000,005	3,105,536	3.1	Web connectivity matrix

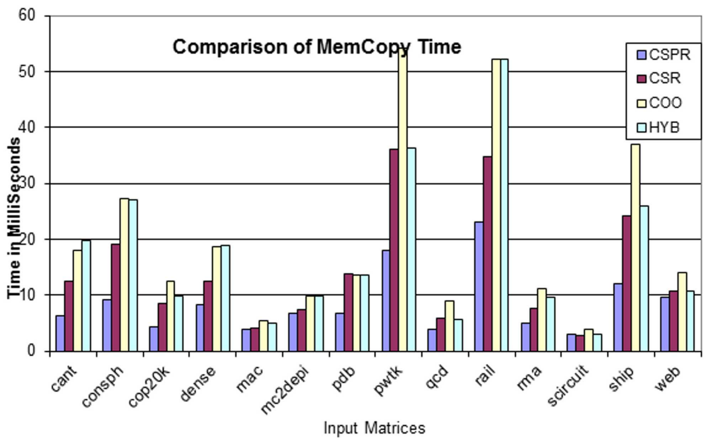


Fig. 7.1: Comparison of memCopy time between CPU and GPU

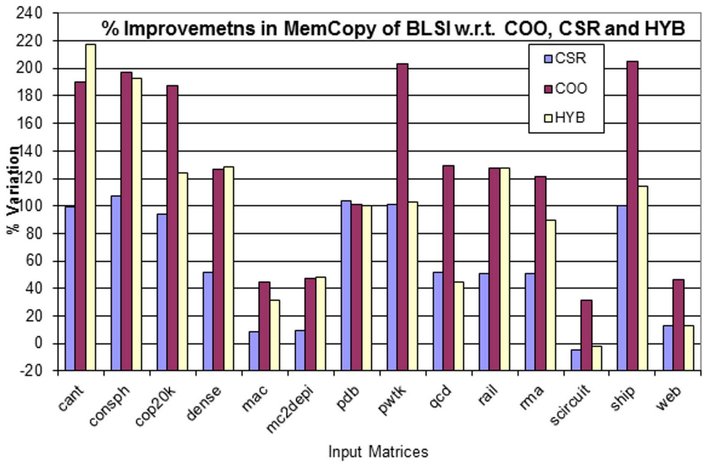


Fig. 7.2: Percentage of variation in Memory Transfer Time between CPU and GPU

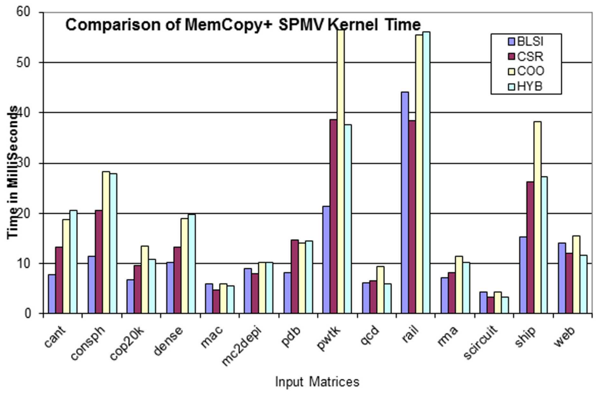


Fig. 7.3: Performance comparison in time that includes memCopy time and kernel execution time

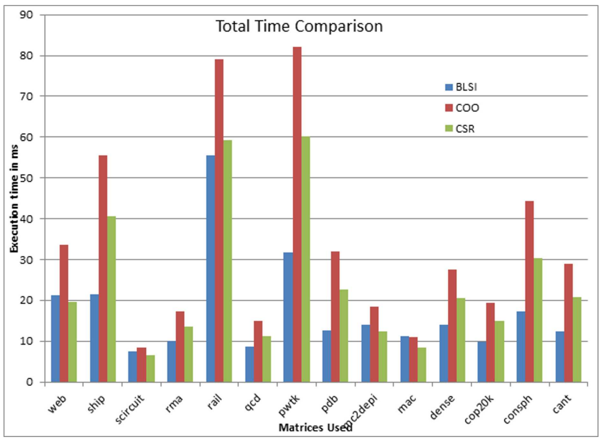


Fig. 7.4: Performance comparison of different formats by considering the total time as time that includes data organization time, memCopy time and kernel execution time

than the BLSI format. These graphs show that how the memory bound computations affect the performances of GPU.

Performance in terms of GFlops and GBytes by considering only kernel execution: The performance in GFlops is computed using only the kernel time. As BLSI do not improve the kernel time of SPMV than the HYB etc., the GFlops computed considering the kernel time is less for BLSI format. BLSI improves the overall time of the SPMV computation at the cost of increased kernel execution time. GFlops graph is shown in Figure 7.5 and bandwidth measurement in GBytes is shown in Figure 7.6. These graphs show the performances of various formats on the GPU.

8. Conclusions. Our experiments have shown that single indexed sparse matrix representation can give substantial improvement in performance while considering the total time of SPMV computation. Total time includes the time for the main operations involved in SPMV, i.e. time of data organization, time of memory transfer and time of SPMV computation on the GPU. The improvement in performance is because of reducing the complexity of data organization and reducing the data to be transferred at the cost of GPU execution time. The improvements shown are not negligible and even for an iterative solution the improvement is beyond the multiple of number of iterations.

The idea of reducing the memory transfer overhead and data organization overhead by using a new data structure for the sparse matrix is novel and it can be improved further. The SPMV performance on the GPU can further be improved by using various optimizations like parallel reduction for row wise sum calculation to mention a few. BLSI or any other formats performance is determined with respect to the input matrix

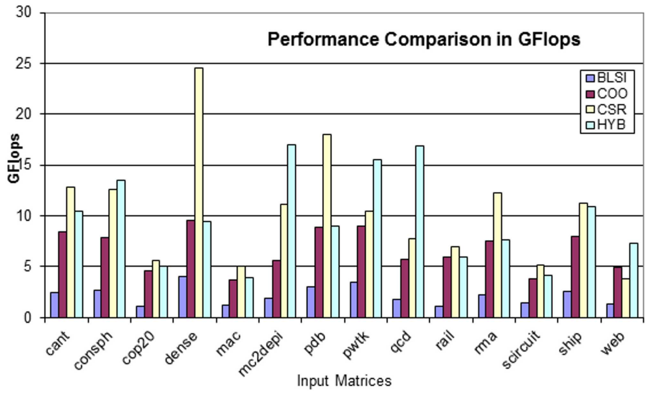


Fig. 7.5: Performance comparison in GFlops considering only kernel execution time of SPMV

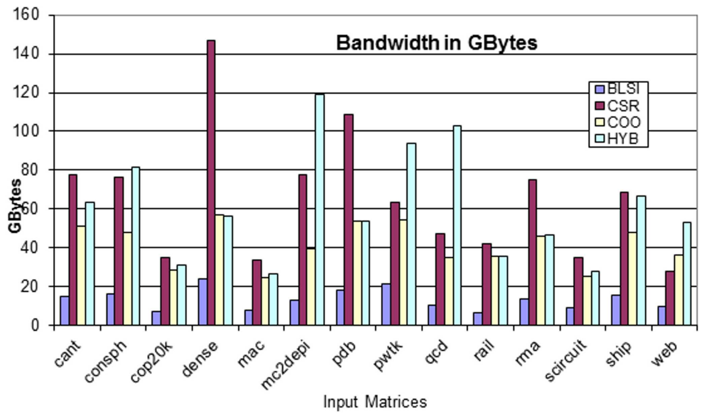


Fig. 7.6: Comparison of bandwidth in GBytes for the different formats considered

sparseness. BLSI kernel performance on GPU is less compared to other formats when the matrix has few rows with large number of elements. An automated tool that can suggest a best data structure from the existing sparse formats based on the input matrix properties like number of elements per row is an on-going work. This can be integrated at runtime so that on the fly data structure to be used can be decided based on the input matrix and also architecture of the device to give the best performance for the SPMV computation.

REFERENCES

[1] *http://developer.nvidia.com/*

[2] Sanders, J and Edward, K. ,*CUDA by example: an introduction to genral purpose GPU programmingCompanion*, Addison-Wesley Professional, 2010,PP: 312.

[3] Ravi Reddy, Malexey Lastovetsky and Pedro Alonso, *HETEROPBLAS: a set of parallel basic linear algebra subprograms optimized for hetergeneous computational clusters Companion*, Scalable Computing: Practice and Experience,10(2)(2009), pp. 201–216.

[4] Xingfu Wu, Valerie Taylor, Charles Lively and Smeh Sharka*Performance analysis and optimization of parallel scientific applications on CMP clusters Companion*, Scalable Computing: Practice and Experience, 10(1)(2009), pp. 61–74.

[5] Vuduc, W. R., Chandramowlishwaran, A., Choi, J., Guney, M., and Shringarpure, A. *On the limits of GPU acceleration Companion*, in USENIX conference on hot topics in parallelism-HotPar-10. Berkeley, CA, USA, 2010, pp. 13–19.

[6] Lee, V., Kim, C., Chhugani, J., Deisher, M., Kim, D., Nguyen, A., Satish, M., Smelyanskiy, M., Chennupaty, S., and Hammarlund, P. *Debunking the 100x GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU Companion*,in 37th Annual International Symposium on Computer Architecture, Saint-Malo, France, 2010, pp. 451–460

[7] Gregg, C., and Kim, H. *Where is the data? why you cannot debate CPU vs. GPU performance without the answer Companion*,

in 11th IEEE International Symposium on Performance Analysis of Systems and Software- ISPASS-11. Austin, Texas, 2011, pp. 134–144.

[8] Bauer, H., Cook, H., and Khailany, B. *CudaDMA: optimizing GPU memory bandwidth via warp specialization Companion*, in International Conference on High Performance Computing, Networking and Storage Analysis. (SC-11), Seattle, Washington, USA, 2011, Article 12.

[9] Thomas, B. J, Prabhu, J.P., Jablin, J.A., Johnson, P.N., Beard, R.S., and August, I.B. *Automatic CPU-GPU communication management and optimization Companion*,in 32nd ACM SIGPLAN conference on Programming language design and implementation (PLDI-11), san Jose, California, USA, 2011, pp. 142–151.

[10] Bell, N., and Garland, M. *Implementing sparse matrix-vector multiplication on throughput-oriented processors Companion*,in International Conference on High Performance Computing, Networking and Storage Analysis (SC-09), Portland, OR, 2009, Article 18.

[11] Blelloch, G. E., Heroux, M. A., and Zagha, M. *Segmented operations for sparse matrix computations on vector multiprocessors Companion*,Technical Report, Department of Computer Science, Carnegie Mellon University (CMU), Pittsburgh, PA, USA, CMU-CS-93-173.

[12] Jee, W. C., Singh, A., and Vuduc, W.R. *Model-driven autotuning of sparse matrix-vector multiply on GPUs Companion*,in 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP-10), Bangalore, India, pp. 115–126.

[13] Yang, X., Parthasarathy, S., and Sadayappan, P. *Fast sparse matrix-vector multiplication on GPUs: implications for graph mining Companion*, Proceedings of the VLDB Endowment VLDB Endowment Hompagearchive, 4(4)(2011), pp. 231–242.

[14] Monakov, A., and Arutyun, A. *Implementing blocked sparse matrix-vector multiplication on NVIDIA GPUs Companion*,in 9th International Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS-09), Samos, Greece, 2009, pp. 289–297.

[15] Monakov, A., Avetisyan, A., and Lokhmotov, A *Automatically tuning sparse matrix-vector multiplication for GPU architectures Companion*, in 5th international conference on High Performance Embedded Architectures and Compilers (HiPEAC-10), Pisa, Italy, 2010, pp. 11–125.

[16] Vazquez, F.,Ortega, G., Fernndez, J.J and Garzn, E.M. *Improving the performance of the sparse matrix vector product with GPUs Companion*,in 10th IEEE International Conference on Computer and Information Technology (CIT-10), Bradford, UK, 2010, pp. 1146–1151.

[17] Maringanti, A., Athavale, V., and Patkar, S. *Acceleration of conjugate gradient method for circuit simulation using CUDA Companion*,in 16th annual IEEE International Conference on High Performance Computing (HiPC 2009), Kochi, India, 2009, pp. 438-444.

[18] Mateti, K.K., and Kothapalli, K. *Accelerating sparse matrix vector multiplication in iterative methods using GPU Companion*,in International Conference on Parallel Processing (ICPP-11), Taipei, Taiwan, 2011, pp. 612–621

[19] Dziekonski, A., Lamecki, A., and Mrozowski, M. *A memory efficient and fast sparse matrix vector product on a GPU Companion*,Progress in Electromagnetic Resaerch, 116 (2011), pp. 49–63.

[20] Neelima, B., and Prakash, S. R. *Effective sparse matrix representaiton for the GPU architectures Companion*,International Journal of Computer Sceince, Engineering and Applications, 2(2),(2012), pp. 151–165.

[21] Im, E. J., Yelick, K., and Vuduc, W. R. *Sparsity: optimization framework for sparse matrix kernels Companion*,International Journal of High Performance Computing Applications, Sage Publications, CA, 18(1), (2004), pp. 135–158.

[22] Vuduc, W. R., and Hyun-Jin, M. *Fast sparse matrix vector multiplication by exploiting variable block structure Companion*, in High- Performance Computing and Communications (HPCC-05), Sorrento, Italy, 2005, pp. 807–816.

[23] Vuduc, W.R. *Automatic performance tuning of sparse matrix kernels Companion*, Ph. D. Thesis, University of California, Berkeley, CA, USA, 2003.

[24] Vuduc, W.R., James, W. D., and Katherine, A. Y. *OSKI: a library of automatically tuned sparse matrix kernels Companion*, J. Phys.: Conf. Series, IOP Science, 16 (2005), pp. 521–530.

[25] Lee, B. C., Vuduc, W. R., Demmel, J., and Yelick, K. *Performance models for evaluation and automatic tuning of symmetric sparse matrix-vector multiply Companion*,in International Conference on Parallel Processing (ICPP04), Montreal, Quebec, Canada, 2004, pp. 169–176.

[26] Williams, S., Oliker, L., Vuduc, W.R., Shalf, J., Yelick, K ., and Demmel, J. *Optimization of sparse matrix-vector multiplication on emerging multicore platforms Companion*,in International Conference on High Performance Computing, Networking and Storage Analysis (SC-07), Reno, Nevada, 2007, Article 38, 12 pages.

[27] Williams, S., Shalf, J., Oliker, L., Kamil, S., Husbands, P., and Yelick, K. *Scientific computing kernels on the Cell processor Companion*, International Journal of Parallel Programming, Kluwer Academic Publishers Norwell, MA, USA, 35(3), (2007), pp. 263–298.

[28] Brahme, D., Mishra, R.B., and Barve, A. *Parallel sparse matrix vector multiplication using greedy extraction of boxes Companion*,in International Conference on High Performance Computing (HiPC-11), Goa, India, 2011, pp. 1–10.

[29] Zein, H.E.A., and Rendell, P. A. *From sparse matrix to optimal GPU CUDA sparse matrix vector product implementation. Companion*, in 10th IEEE/ ACM International Conference on Cluster, Cloud and Grid Computing, Melbourne, Victoria, Australia, 2010, pp. 808–813

[30] Wang, W., Xu, X., Zhao, W., Zhang, Y., and He, S. *Optimizing sparse matrix-vector multiplication on CUDA Companion*,in 2nd International conference on Education Technology and Computer (ICETC-10), Shanghai, 2010, pp. 109–113

[31] Baskaran, M.M., and Bordawekar, R. *Optimizing sparse matrix- vector multiplication on GPU Companion*, IBM Research, 24704 (2008).

[32] *Cusp-library, http://code.google.com/p/cusp-library/*

[33] *http://www.cise.ufl.edu/research/sparse/matrices/*

Edited by: Marcin Paprzycki
Received: May 1, 2012
Accepted: June 15, 2012

Scalable Computing: Practice and Experience
Volume 13, Number 2, pp. 173–186. http://www.scpe.org
ISSN 1895-1767
© 2012 SCPE

DYNAMIC DISTRIBUTED PROGRAMS CONTROL BASED ON GLOBAL PROGRAM STATES MONITORING

J. BORKOWSKI[†] AND M. TUDRUJ[†‡]

Abstract. The paper concerns designing distributed program execution control based on global application states monitoring in the presence of a dynamic number of processes and threads. Global program execution control is based on application states monitoring at the level of processes/threads in clusters of multi–core processors. A special control infrastructure is proposed based on synchronizers, which collect state information from processes and threads, detect strongly consistent application global states, evaluate control predicates and send respective control signals. An algorithm for detection of strongly consistent global states for a variable number of processes/threads is presented.

1. Introduction. Writing distributed programs with complicated global control structures for execution in clusters of processors is easier when some support for automatic monitoring and handling of global application states is provided in the run time system. Commercial distributed systems do yet not support this idea. Hence, the program execution control based on the monitoring of global application states has to be programmed by programmers from scratch. It requires tedious programming of the collection of constituent local states and designing out of them global states which had happened in parallel in distributed processes of an application. Programming these aspects of program execution control is complicated and takes much of programmer's time. Besides it is error prone. This paper relates to an original distributed program design framework PEGASUS (Program Execution Governed by Asynchronous SUpervision of States) [6, 24], which assumes a built in support for handling the local and global application states and automated design of program execution control based on global states monitoring.

Some introductory research results on the use of the global application states monitoring in the design of execution control of distributed programs can be found in literature. Linda environment [1] provides a common global tupple space for the exchange of global control information and supports primitives for writing and reading in it. Some solutions leading to the design of global control for interactive software components can be found in coordination languages. Manifold and Reo environments [2] have ben provided with primitives for inclusion of communicating software components into coordinated structures. Nevertheless, no notion of a global state has been used in these systems. Global application states were basic concepts in the Meta system [3], which enabled designing distributed programs based on communicating components. In the Meta system, application processes were able to send messages on their local states on which consistent global states were generated and the respective global predicates were evaluated. In Meta, complicated formalism based on guards was used to express control based on global states. An attempt to simplify this formalism was undertaken in the Lomita language [3]. But it suffered of inefficiency due to very costly ordered state message broadcasts. Global control constructs for the OCCAM language were proposed in [4] with an implementation based on replication of global state variables. The first infrastructure for the run–time supported design of the asynchronous global execution control of distributed programs in C language based on monitoring of global application states was implemented in a graphical parallel program design system PS–GRADE [5].

The features of the PS–GRADE and other distributed program design frameworks [9] have been substantially extended in PEGASUS. The main extension are the graphically supported global high level control flow constructs in which the flow of control at the level of the distributed program components depends on the predicates computed on global application states. In this respect, the PS–GRADE did not provide any mechanisms for structural control flow steering by global application states while taking care of the asynchronous control of the internal process behaviour. Another important extension is the global program execution control exercised at the thread level, important for the evolved software design for multicore processors. One more extension is the separation of the communication frameworks for maintaining the control data from the communication environment for computational data. This separation is on one hand logical and on the other hand physical since separate networks are assumed. Such logical and hardware support strongly improves control design efficiency as has been already discovered by simulation experiments for PS–GRADE [21].

Current implementation of the PEGASUS framework assumes that the number of processes and threads

[†]Polish–Japanese Institute of Information Technology, 86 Koszykowa Str., 02–008 Warsaw, Poland (janb@pjwstk.edu.pl)

[‡]Institute of Computer Science, Polish Academy of Sciences, Jana Kazimierza 5, 01–248 Warsaw, Poland (tudruj@pjwstk.edu.pl)

whose local states contribute to the definition of the program global consistent states, is known at the application program compile time. In the research reported in this paper, the number of threads and processes taken into account is assumed to be variable and is set dynamically at program run time. The variable number of processes has been considered in [7, 8, 9, 10, 11], however only for the group data communication control. With the new assumptions, in the dynamic distributed programs designed under the PEGASUS framework, the automatically constructed global states based on local states of a variable number of threads or processes define the control flow and the functional features of programs. These features extend the functionality of the PEGASUS framework towards more flexible control constructs, however at the price of new algorithms for the construction of strongly consistent global states and more specific methods for state messages sending and signal reception in application program components. Both these problems are presented and discussed in this paper.

A short presentation of this research has been published in [25]. The present paper adds many details and includes the following relevant material:

- dynamic SCGS detection algorithm described in detail with a pseudo–code,
- an application example,
- a description of the PEGASUS system as an example of an environment where the dynamic SCGS detection can be applied.

The paper consists of six parts. In the first part, the application program execution control based on global states monitoring is explained. The second part, presents the related research on the dynamic process membership in program control. The third part presents the algorithm for the detection of global application states in the presence of a variable number of processes/threads. The fourth part shortly describes the graphical design of application programs in the PEGASUS framework. The fifth part discusses global program execution control based on dynamic process groups. The sixth part outlines a relevant example with a variable number of processes.

2. Global states monitoring for application control. An analysis of existing parallel application control methods e.g. [12, 13, 14], has motivated us to have a global view of a running distributed application state used when defining distributed program execution control. Such control should be based on global program execution control high–level primitives. To increase the program code clarity, the code sections responsible for program execution control should be separated from the code sections responsible for computations. To increase the distributed program performance, the synchronization and communication of the control data and steering orders should be implemented without passive waiting.

Initial ideas on global distributed programs control [4, 15] have been rectified into program execution control methods corresponding to current parallel and distributed program semantics, which intensively used a precisely defined notion of the application global state [16]. The new program control model and its theoretical background were implemented inside a parallel program graphical design environment P–GRADE giving an extended version of it, called PS–GRADE [5]. In PS–GRADE, special control processes called synchronizers were able to receive local state reports from processes of a running application. Based on the local state information, the synchronizers were able to re–construct and monitor the global states of an application for defined sets of relevant constituent processes. Among different kinds of global states, the most interesting were such for which the synchronizers and the application program programmer were sure that the states occurred in distributed processes for sure in parallel, with the occurrence not weakened by any knowledge based on partial orders. Global application states discovered by synchronizers with such certainty are called Strongly Consistent Global States (SCGS) [19]. A SCGS can require local states from all processes of a distributed application or of a subset of them. Based on that, we speak about absolutely global or regional strongly consistent global states. A programmer can also consider states which do not correspond to concurrent local states without any concurrency requirements. They are called Observed States. For observed states no SCGS detection algorithms are performed and local states are directly used for application control definition. The PS–GRADE system run–time framework, working on–line in parallel with the application, was providing the infrastructure for sending process local states, reception of the state messages by synchronizers, reconstruction of different kinds of global application states including SCGSs and using them to define the execution control of the application. The sequence of messages the synchronizers were receiving might not be the same as the sequence of events in reality, because of delays in message construction, transfer and processing which could happen. Therefore, the received local state messages were only used as input data for consistent state construction algorithms [18]. For our SCGS generation algorithms, the application processes were sending local state messages accompanied by real–time event timestamps obtained with the use of partially synchronized local processor clocks globally

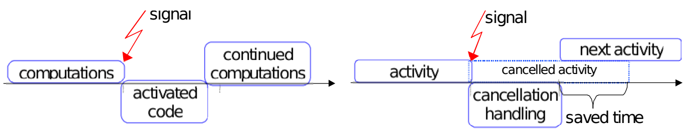


Fig. 2.1: Code activation (left) and cancellation (right) by control signals

synchronized with a known accuracy [16]. The SCGSs based on such local state messages can be constructed in O(E NlogN) time, where N is the number of processes and E is the number of events per process. The complexity of the algorithm with real–time event timestamps is lower than that of the algorithms, which use logical vector clocks since the later leads to SCGS detection with the exponential complexity. By using networks of parallel synchronizers, a distributed hierarchical SCGS detection can be organized, which is speeding up the SCGS detection for complicated state cases [17].

An SCGS is a state, which has occurred for sure in all involved processes, in difference to potential global states obtained with the use of logical clocks. Timestamps defined with a known accuracy requires a processor clocks synchronization facilities installed in the executive distributed system.

The clock synchronization needed here can be obtained in many ways, depending on the assumed accuracy and budget. The least costly and the cheapest is the Network Time Protocol which can reduce the clock skew to tens of μs. The more costly is the Precision Time Protocol, which supported by a dedicated time messages transmission network can provide the clock skew below 1 μs.

Control predicates used for definition of process execution control represent control conditions, which are checked when a given global state is reached. The predicates are encapsulated in synchronizers as specially structured pieces of code, which are defined by a programmer as parts of the application. The predicates can be defined based on SCGSs (global or regional) or observed states. For SCGSs, the predicates are checked on each detected SCGS state. If a predicate is fulfilled, the Unix–type control signals are sent to selected processes. The signals can activate designated code sections in target processes immediately upon their arrival, suspending current computations. Alternatively, they can cancel current computations, making processes to proceed immediately to further parts of their algorithms, see Fig. 2.1.

If the signals are sent between a synchronizer and processes located on different processors, the signals are transformed into control messages when they leave the processor of the synchronizer. The messages are sent by the use of a message passing library. The messages are asynchronously received at the target process processor and then transformed into classic Unix signals which are delivered asynchronously. With the use of asynchronous control signals, a process does not need to stop and wait passively for control orders sent by synchronizers.

In PS–GRADE, the set of parallel application processes taken into account for the monitoring of global application states was defined statically by a programmer. This limitation can be removed in the new enhanced PEGASUS system. Under PEGASUS, each process usually contains many parallel threads which can be monitored and controlled by synchronizers through global predicates, in a similar way as processes [17]. It is a quite common programming practice to create and destroy the threads when necessary. Therefore, the global control system should enable to deal with dynamically changing set of monitored entities (threads or processes). In this paper, we examine methods for monitoring global states of dynamically changing sets of threads/processes and methods for organizing the respective application control.

3. Related research. The problem of cooperation between processes in a group when the processes can join and leave the group has been analyzed in the literature. Several authors studied such cooperation using broadcast as the main communication primitive.

A good insight to the problem in presented in [7] and [8]. There, the notion of views was defined. A view stands for an interval when the group membership is constant. In a single view classic (static) algorithms can be used to implement broadcasts. The difficulty has been moved largely to view maintenance — creation of new views and installation of new views in processes. A similar approach has been taken in the Totem system [9]. A dedicated service was created there to monitor and maintain the membership.

The idea, that processes see the same membership within a specific view (time period) and that they deliver the same set of messages in each view in the same relative order got its name as “view synchrony” and has been examined by other authors, e.g. [11]. View synchrony uses broadcast communication and orders events logically (before/after) ignoring actual (wall clock) time relations. The real time is used only to detect dormant members, e.g. as in [20], where assumptions on maximal token transfer time exist.

Our case differs from the cases analyzed in the literature in a few important aspects.

- The communication we need is of many–to–one type (processes reporting to a synchronizer) and one–to–many type (a synchronizer sending control signals to processes). It is not many–to–many communication — processes do not need to know about each other.
- We use Strongly Consistent Global States and real–time timestamps. This makes the coordination between processes much different than in the case of logical time.
- The control mechanism in our system is based on consistent global state monitoring — we must be able to construct consistent global states with changing process membership.
- The emphasis in our system is not on communication (sending messages with data), but on control (sending commands and states). The impact of membership changes should be analyzed from another point of view.

4. SCGS detection.

The Strongly Consistent Global States (SCGS) were introduced in [19]. They are analogous to the Consistent Global States (CGS) as defined in e.g. [18]. A CGS is a set of local process states, one state from each process, with the property that the states are pairwise concurrent. The concurrency is defined with the happened–before relation and logical vector clocks can trace it. In SCGS the state concurrency is defined with the help of real–time (wall clock) timestamps. To be able to use real time clock the processes must have access to a global shared clock or their local clocks must be synchronized. It is assumed that the clocks used by the processes are not ideally synchronous and that the upper bound of the clock skew is known. In such a situation, an event can be depicted on a time axis as an interval rather than a point. An event occurs momentarily of course, but we are able to pinpoint the time of its occurrence due to not perfect clock synchronization. Instead, we determine an interval within which we know the event has occurred. Fig. 4.1 illustrates how the events are marked on the time axis and how SCGSs are constructed from concurrent local states.

The SCGSs (unlike CGSs) occur sequentially, one at a time. The number of them is linearly proportional to the number of events. What follows is that the algorithm which constructs SCGSs is much less complex than algorithms for CGS lattice construction. To be able to explain the proposed extensions to the SCGS algorithm, we will recall the idea of the standard SCGS algorithm first [16].

We will formulate the SCGS construction algorithm using the diagram from Fig. 4.1. The local process states for each process are depicted as a sequence of segments (think solid lines), each segment stands for a local process state duration. When the segments are projected onto the bottom line, then the rectangles with names S1,S2,S3,S4 mark the areas where the projections have non–empty intersections. These are the SCGSs. The algorithm has to find them. The segment positions in each sequence are sorted and the segments are disjoint within a single sequence. For now we assume, that intervals between consecutive events at one process are longer than 2ϵ, where ϵ is the clock synchronization accuracy. The notation is as follows:

SEQ_i — segment sequence for process i,

CS — the currently examined set of segments containing one segment from each sequence,

s_i — segment from SEQ_i in CS,

$S(s)$ — segment s start position,

$T(s)$ — segment s terminate position.

$next(s_i)$ — the next segment after s_i in SEQ_i

The intersection is non–empty if $i, j = 1..n : S(s_i) < T(s_j)$, which can be simplified as $max_i = 1 \ldots N(S(s_i)) < min_i = 1 \ldots N(T(s_i))$ (condition C1)

Initially CS contains the first segment from each sequence. If the condition is not met, then there exist k and l, such that $S(s_k) \geq T(s_l)$. Assuming that we have checked all segments lying before the ones currently in CS (which is true initially) there is no point in an attempt to decrease $S(s_k)$, but we can take the next segment in SEQ_l to increase $T(s_l)$. In such a way we proceed forward moving by one segment from one sequence at a time, preserving the assumption. If a nonempty intersection is found, we have to start the search over. There exists k, such that $T(s_k) = T(CS)$. It is enough to notice, that the next non–empty intersection cannot start earlier than $S(next(s_k))$. So we take the next segment from SEQk and run the procedure further, again with the assumption valid. By induction, no non–empty intersection will be missed and all segments will be checked.

When we allow short intervals between events on a single process, we can have a situation as on the left–hand side of Fig. 4.2. There is a “negative length” segment between events e1 and e2. It should be interpreted

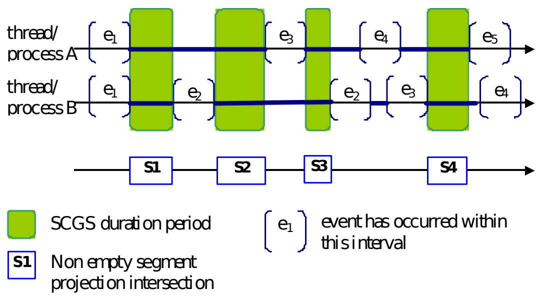

thread/ process A
thread/ process B
e1
e2
e3
e4
e5
S1
S2
S3
S4
SCGS duration period
Non empty segment projection intersection
event has occurred within this interval

Fig. 4.1: SCGSs detection

that we do not know the exact start/end points of the segment, yet we know there is one here. For P1 in a state between events e1 and e2 and P2 between f1 and f2 condition C1 is false. However, we do have a nonempty intersection here. Only we cannot precisely point its position. The right–hand side of Fig. 4.2 shows a similar situation, but this time the intersection is empty. We formulate the proper condition as follows:

$$(C1) \vee (\exists i : S(s_i) = max(S(s_k)) \wedge T(s_i) = min(T(s_k)) \wedge$$
$$max2(S(s_k)) < T(s_i) \wedge min2(T(s_k)) > S(s_i))$$

where $k = 1 \ldots N$ and $min2()$ and $max2()$ give the second minimal and maximal value, respectively. The condition says, that if we have a "negative length" segment, then it must be fully contained in other segments.

To speed up the condition evaluation, tuples $\langle S(s_i), i\rangle$ and $\langle T(s_i), i, ptr_i\rangle$ should be kept in priority queues with the first element as the key – element insertion cost is logN, accessing min/max is done at a constant cost. Component ptri points to the corresponding tuple (start of segment i) in the first queue. If we notice, that $T(SC) = min_{i=1 \ldots N}(T(s_i))$, then the search for $k : T(s_k) = T(SC)$ can be accelerated by using one of the introduced queues. Operations $min2$, $max2$ and removing an element pointed by ptr_i can be done at cost logN. The total cost of the algorithm is O(E NlogN).

4.1. SCGS in dynamic groups. The described above SCGS detection algorithm assumes a constant number of monitored processes. Each report coming to a synchronizer contains information about an event in a known process. To allow a process to join and leave the monitored (and therefore controlled) group, a synchronizer must be informed about the process group membership changes. We introduce two new message types for that purpose. A join message is used by a new process to tell a synchronizer that it is joining the controlled group, while the leave message terminates the process group membership. The assumption that process clocks are synchronized within a known accuracy must stay true for the joining (and of course the leaving) processes. With this assumption the new messages can be ordered on a time axis using the same rules as standard event messages. As described above, the standard SCGS detection algorithm hops from one event to the nearest one. With dynamic process membership, it is enough to verify if a join or leave message with a timestamp positioned between the current event and the nearest one has been received. A join event should be then processed as follows: increment the number of monitored process $N := N + 1$, rebuild all data structures to reflect this change, set the new process state as reported within the join message. A leave message processing should decrease the number of monitored processes $N := N - 1$ and all the involved data structures should be rebuilt accordingly, removing the elements corresponding to the process that has left.

Fig. 4.3 shows a situation when the ideal scenario fails. A process join message e_{join} from process C contains a timestamp which should position this event within SCGS S1 and S1 should include three processes. However, the message e_{join} arrives with a delay, after S1 has been detected for two processes only. Without additional assumptions the network delays cannot be anticipated and one should be aware that process joining

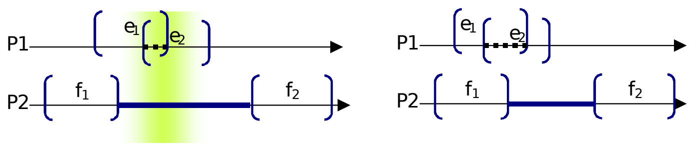


Fig. 4.2: Short–lasting local states can lead to an SCGS (left) or not (right)

can be noticed by a synchronizer with a delay.

Fortunately, in environments where the network delays are bounded a better solution is possible. In [21] a SCGS detection algorithm for bounded message transfer times is presented (the UT algorithm). The algorithm postpones processing of the received event reports to make sure that all delayed reports arrive first. The postponing time is computed according the bounded message transfer time. With this algorithm, the detection of S1 in Fig. 4.3 would be postponed until e_{join} arrives. Then, Process C would be added to S1 according to the timestamp value in e_{join}. The UT algorithm has an obvious drawback — all SCGS detections must be delayed by the postponing time (proportional to the bound of the message transfer time).

Process leave scenario is shown in Fig. 4.4. The message e_{leave} arrives just any normal event report with a timestamp. When S2 is detected, e_{leave} is treated as a normal event report and marks S2 termination. In the next step, e_{leave} is recognized as a leave message. Information about Process C is deleted from the algorithm data structures, but prior to that e_{leave} timestamp is copied to the remaining process to mark the beginning of the next SCGS. The next SCGS S3 contains only processes A and B.

Below, we present the code of an SCGS detection algorithm able to deal with dynamic thread/process membership. The algorithm is an enhanced version of the "T" algorithm from [21] — processing of "join" and "leave" messages has been added. "T" is the simplest SCGS algorithm variant and its presented enhancement permits that the states of joining processes can be taken into account with some delay, as has been explained already. For simplicity, the presented code does not perform data structure compacting — the processes that have left remain as empty entries. This has the advantage that the array indices used as process identifiers are unique (new processes get new indices) and the algorithm presentation is as clear as possible. When necessary, indirect indices can be introduced, i.e. instead if index i in P_i one should use ind_i and manage the ind array appropriately. Such a solution is conceptually straightforward, therefore we will not delve into its technical details.

ALGORITHM 1. SCGC detection using terminated local states

Symbols and data structures:

ϵ_i	— quality of clock synchronization, for P_i's clock c_i and the master clock $m : \|c_i - m\| < \epsilon_i$
P_i	— process number $i, i = 1 \dots N$
$e.C$	— the timestamp attached to event e, a positive number. The value of ϵ_i is known by the monitor and it is the monitor task to convert scalar timestamps into intervals: $e.C \to \langle e.C - \epsilon_i, e.C + \epsilon_i \rangle = \langle e.C1, e.C2 \rangle$
s_i	— currently processed local state of Pi
$s.S$	— event that starts state s
$s.T$	— event that terminates state s

Necessary data structures are as follows:

- FIFO queues $Q_i, i = 1 \dots N$. Q_i holds messages (events) from P_i. Q_i.first() is the termination event of the currently considered P_i state. The following operations are defined on queues, each with the cost of O(1): Q.first() — the first element in the queue. If the queue is empty, then the method blocks until an element is available; Q.append(s) — puts state s into the queue; Q.remove() — discards the first element; Q.empty() — TRUE iff the queue is empty.
- Array $S_{1 \dots N}$ (Starting) contains events. S_i is the starting event of the currently considered P_i state: $S_i = s_i.S$. The algorithm checks if local states $s_i : s_i.S = S_i, s_i.T = Q_i$.first(), $i = 1 \dots N$, are pairwise concurrent.
- Priority queue $maxS$ holds pairs $\langle S_i.C + \epsilon_i, i \rangle$, the first pair component is used as the sort key.

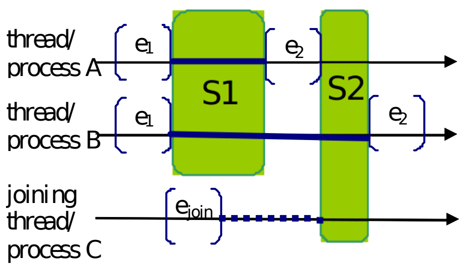


Fig. 4.3: A delayed join message

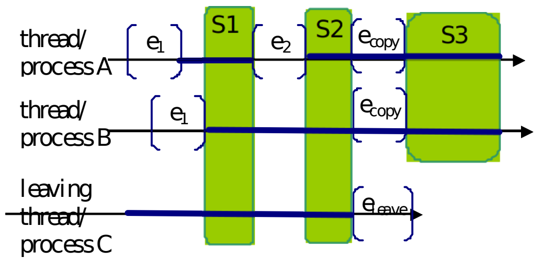


Fig. 4.4: Processing of a process leave message

- Priority queue $minT$ contains records $\langle Q_i.first().C - \epsilon_i, i, ptr\rangle$, the first component is used as the key. A component ptr points to a corresponding record in $maxS$ (record with the same i component). This queue block the $min()$ operation until the number of entries is equal $N - L$.
- The following operations are defined on priority queues:
  - insert(t,i) : pointer — adds a new record in O(logN), returns a direct pointer to the new element
  - max() — returns a record with the maximal key in O(1)
  - min() — returns a record with the minimal key in O(1)
  - max2() — returns a record with the second maximal key in O(1);
  - min2() — returns a record with the second minimal key in O(1);
  - removemin() — removes a record with the minimal key in O(logN);
  - remove(ptr) — removes a record pointed by ptr in O(logN).
- Array $maxSPtr_{1...N}$, $maxSPtr_i$ points to record $\langle t, i, ptr\rangle$ in maxS. It is used by $minT$.insert() to get a pointer to a corresponding record in $maxS$.
- Initially $\forall_{i=1...N} : Q_i.empty() =$ TRUE; $\forall_{i=1...N} : S_i = e0$, where $e0.C = 0$; $maxS$ contains $\langle 0, i, ptr\rangle$, $i = 1 \dots N$; $maxSPtr_i$ points to record $\langle t, i, ptr\rangle$ in $maxS, i = 1 \dots N$; $minT$.empty() = TRUE.

The algorithm is as follows:

```
on reception of the message $e$ (including a leave message) from $P_i$:
    if $Q_i$.empty() then {
        $Q_i$.append($e$)
        $minT$.insert($e.C - \epsilon_i$, $i$, $maxSPtr_i$)
    } else
        $Q_i$.append($e$)
    if "monitor was suspended waiting for an event from $P_i$" then
        "resume monitor"

on reception of a join message $j$ from $P_{N+1}$:
    $N = N + 1$
    $S_N = j$
    $maxSPtr_N = maxS$.insert($\langle j, j.C + \epsilon_i, N\rangle$)

main loop:
    loop
        $\langle t, i, ptr\rangle = minT$.min() //blocks until minT has $N - L$ elements
        $min = t$
        $\langle max, j\rangle =$ maxS.max()
        if $min > max$ then {
            /* SCGS found, SCGS=$\langle s_1, \dots, s_N\rangle$, where $s_i$ is a local state of $P_i$
            between events $S_i$ and $Q_i$.first() */
            if "it is a termination SCGS" then
                loopexit
        } else if $i == j$ then {
            /* 2 events from the same process */
            $\langle min2, i, ptr\rangle = minT$.min2()
            $\langle max2, j\rangle = maxS$.max2()
            if ($max2 < min$ AND $min2 > max$) then {
                /* SCGS found */
                if “it is a termination SCGS” then
                    loopexit
            }
        }
        if “$Q_i$.first() is a leave event” then {
            $L = L + 1$
            $minT$.removemin()
            $maxS$.remove($ptr$)
            $Q_i$.remove()
        } else {
            /* search further, the state which terminates first is blocking */
            $minT$.removemin()
            $maxS$.remove($ptr$)
            $maxSPtr_i$= $maxS$.insert($\langle t + 2\epsilon_i, i\rangle$)
            $S_i$=$Q_i$.first()
            $Q_i$.remove()
            $t = Q_i$.first().C // blocks if $Q_i$ is empty
            $minT$.insert($\langle t - \epsilon_i,\ i,\ maxSPtr_i\rangle$)
        }
    endloop
```

5. Process/thread control in PEGASUS. The aim of the PEGASUS framework is to support graphical design of application program execution control governed by global application states. The framework enables a graphical design of the global execution control in distributed programs. In PEGASUS, the program global control design is graphically supported at the program global control flow level and the process/thread level. At both levels graph representation of programs is used.

The program global control flow level is developped using Control Flow Graphical Windows. They enable designing a control flow graph of a distributed program, which shows all control flow dependencies between program blocks, for which the control is governed by application global states. This level enables development of a distributed program in terms of nested high level control constructs acting on program blocks. At the program global control flow level a special kind of the window is additionally used Block Interactions Window, which shows the flow of control data between the blocks specified in graphical representation of the program using control flow graphical windows.

The process and thread level enables designing the structure of internal program code in C/C++ language for terminal program blocks in the control flow graph, i.e. such blocks which will not be further developed to contain nested global high level control constructs. Such terminal blocks correspond to structures of application processes which will be directly assigned to processors. These structures are designed in a special kind of graphical windows which is called Process Structure Window.

The terminal blocks will be converted into source program modules compiled into executable modules after mapping of code blocks to resources using the third graphically supported level of windows which are the Synchronizer Editing Windows and the Thread Block Editing Windows.

At the program global control level, an application program is represented as a control flow graph. It is composed of program blocks (the graph nodes) interconnected by arcs representing the flow of control and the flow of control information between blocks. In general, this graph shows how the flow of control between program blocks depends on global states of the program. This dependence is exercised based on the use of predicates computed on program global states. Nodes of a global control flow graph represent program code blocks, which can contain processes or threads (rectangles annotated with block names), synchronizers (annotated with names) containing predicate blocks (with general predicate specifications), control flow switch nodes which direct the flow of control in the graph accordingly to signals generated on the basis of predicate evaluation and control paths parallelization/synchronization primitives such as PAR, JOIN, BARRIERS, EUREKA, etc. The edges in the graph represent the flow of control between nodes and flow of state messages or signals between predicate

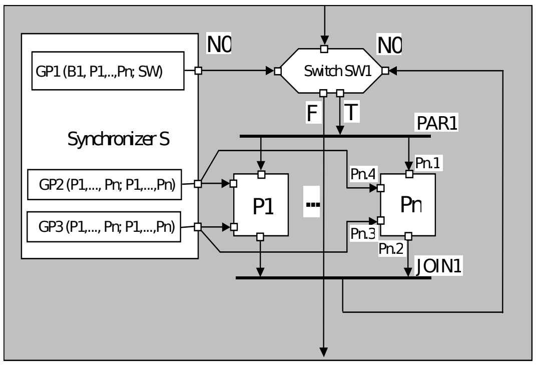


Fig. 5.1: PARALLEL WHILE DO construct with a control flow predicate GP1 and an asynchronous predicate GP2, GP3

blocks, code blocks and switches.

Important elements of the global program control flow graph are synchronizers. The synchronizers are special program blocks which contain control predicate blocks. The synchronizers and the predicates they contain can be assigned to processors or processor cores. A predicate specification includes a predicate name followed by two lists separated by a semicolon: a list of graph blocks (including predicates) whose states are involved in predicate evaluation and a list of signal receiver blocks. We distinguish control flow and asynchronous control predicates. Both can be placed in the same synchronizer. All blocks have communication ports, which are used to draw control data communication paths. Port and port interconnections are automatically drawn by the system if the blocks to connect are specified inside the predicate blocks. However, ports and port connections have to be drawn by a programmer for the control flow which is not specified inside predicates belonging to a program block. In order not to complicate program graph design, the communication concerning computational data is not represented in the program global control flow graphs. The design of the program graph is supported by the replication and nesting facilities. The replication facility enables simplification of the program graph for control constructs, which are acting on the replicated program blocks. The nesting facility enables automatic opening of graphical windows in which nested control constructs can be designed. At block boundary communication ports are automatically generated to enable controlling the interface between the program graph external to the block and the internal graph inside the nested block.

Fig. 5.1 presents an exemplary simple program control flow graph. The graph contains n program blocks Pi governed by a replicated PARALLEL WHILE DO construct. The flow of control is supervised by a global predicate GP3 evaluated on the basis of a global state built of local states of the blocks P1,. . . ,Pn. GP1 sends signals to the switch SW1, which directs the flow of control. SW1 asynchronously receives and stores all signals generated by GP1. When the execution control reaches SW1 it directs the flow of control accordingly to the last signal value received.

Fig. 5.2 presents the control graph nested inside a block Pn. The replicated program blocks Z1,. . . ,Zp are embedded in a PARALLEL DO–UNTIL construct. The flow of control is governed by a global predicate G2 evaluated on the basis of a global state built of local states of Z1,. . . ,Zp. Blocks Z1,. . . ,Zp are asynchronously controlled by the predicate G1, which is evaluated based on the local states of Z1,. . . ,Zp, states of some blocks D1,. . . ,Dv and also signals sent as a result of the predicate GP1 in the synchronizer S. The predicate G1 asynchronously controls Z1,. . . ,Zp by sending signals to them. Z1,. . . ,Zp are additionally directly asynchronously controlled by the predicate GP3 of S.

Before the code of process blocks is designed using the C/C++ language, the process graphs will be first graphically composed of thread blocks and thread level synchronizers in the Process Structure Windows. Fig. 5.3 represents such a window for the replicated Z1,. . . ,Zp

blocks from Fig. 5.2. In this window, we can see the thread level synchronizer TS which asynchronously controls thread blocks

T1 and T2 synchronized by PAR/JOIN constructs. TS contains two predicates TP1 and TP2. TP1 combines local states of T1 with control signals from the predicate G1 of the synchronizer Sn (Fig. 5.2). TP1 controls threads in T1 and sends state to the predicate G1 of Sn. TP2 combines states of T1 and T2 with signals from the predicate GP2 in a common global state. Based on this state, TP2 controls T1 and T2 and

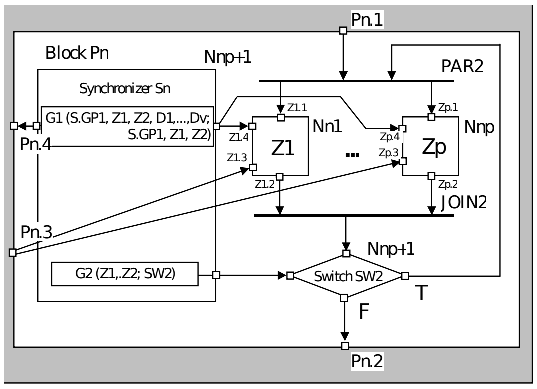


Fig. 5.2: PARALLEL DO–UNTIL construct embedded inside Pn block

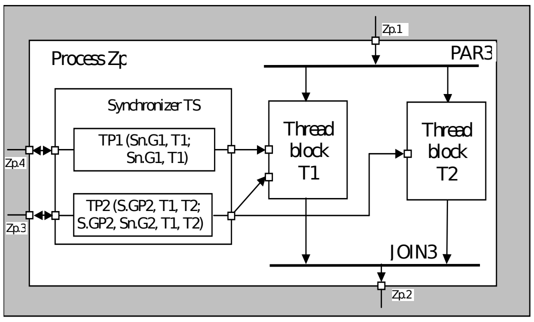


Fig. 5.3: Process Zp structure in the process and threads graphical window

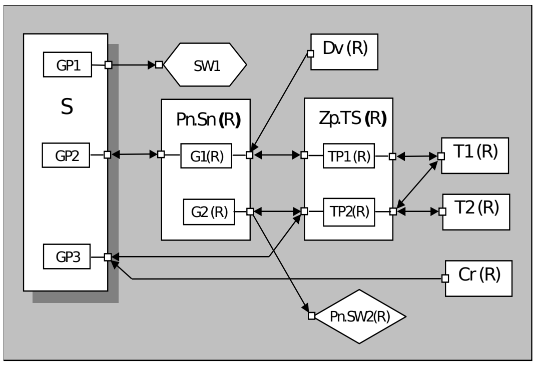


Fig. 5.4: Interactions window of the synchronizer S.

sends states to the predicate G2 of Sn and the predicate GP2 of S.

The PEGASUS framework includes facilities for an easy and convenient representation of the global program execution control. This idea is supported by the Block Interactions Window. It automatically provides a representation of the global control structure in the program related to a selected block of the program control flow graph. This window enables an easy verification of the structure of the global control based on tracing the synchronizers (more exactly their predicates) and thread blocks interactions in the program graph.

The Block Interactions Window shows an interactions graph in respect to global control design for a selected block of the control flow graph. The window is automatically generated as a result of a click on a selected control flow graph block. Fig. 5.4 represents the interactions window of the synchronizer S for the control graphs shown in Fig. 5.1, 5.2 and 5.3. In this window we can trace control links of a selected graph node (the highlighted block

S) with other nodes which co–operate with the selected node. The co–operation is implemented by sending or receiving state messages and signals. Inside synchronizer blocks all relevant predicate blocks are shown with the connections to respective synchronizer ports. The window has special facilities to simplify representation of interactions of the replicated blocks. Program blocks which belong to replicated control constructs are displayed only once with notification (R) – replicated, in the names.

6. Process/threads control in dynamic groups. The processes controlled by a synchronizer do not need to know about each other. Therefore they may not be aware of the changing group membership. However, the actions initiated by a synchronizer can be affected by processes leaving/joining the group. A synchronizer upon analyzing a SCGS can decide to send a control signal to a process. If the process decides to leave the group before receiving the signal, the signal will not be delivered. The simplest option is to ignore such signal. If ignoring could cause problems in an application control scheme, the following possibilities, enumerated below, can be exercised.

- A process should confirm signal reception. Not confirmed signals should be "re–processed" e.g. a predicate should be re–evaluated. This solution requires a timeout — the message transfer time must be bounded — otherwise a synchronizer would not know how long to wait for a confirmation.
- A symmetric solution is a negative confirmation — a subsystem responsible for signal delivery can generate an error message for a synchronizer when a target process does not exist. The message transfer time should be bounded as well, if the synchronizer should be able to "re–process" failed signals — it may need to keep some history information for this purpose for a time period dependant on the message transfer time.
- A process should inform the synchronizer that it plans soon to leave the group. Such processes are not eligible as signal targets. This option assumes that a process knows in advance that it will leave the group (not suitable for dealing with process failures) and again the message transfer time must be bounded, so it is known how early a process must send the information about its approaching group leaving.
- Before sending a signal, a synchronizer should ask a target process if it is ready to receive the signal (not suitable for dealing with process failures). This idea induces a large performance penalty, requiring a round trip communication before each signal.

A control scheme can involve actions on more than a single process at a time. As a practical example of such a situation, we can give various load balancing strategies. In the simplest case, a synchronizer selects an over–loaded and an under–loaded process and signals both of them. The process pair should communicate directly and level their load. If one of the selected processes leaves the group after it has been selected and before a peer process contacts it, a communication problem arises. One process from the pair would stall trying to contact a non-existent process. The following solutions, enumerated below, can be taken into account.

- Do not let processes to communicate directly.
- When communicating with a peer process, always use a timeout mechanism — message transfer time must be bounded.
- A process should inform the synchronizer that it plans soon to leave the group. The details are the same as in point c) in the previous subsection.
- Before sending a signal, a synchronizer should ask a target process if it is ready to receive the signal as in point d) in the previous subsection.

A synchronizer must be able to address a proper process/thread to receive a signal. Some difficulties arise here if we want to guarantee that any newly joined process is distinguished from all processes that have left the group. Two situations impose problems in this area.

- Process A is selected as a signal target, but before the signal reaches it, process A leaves the group and process B joins it. If B gets the same identifier as A, it will receive the signal targeted at A.
- If a predicate maintains historical data about process states, and if process A leaves the group and process B joins it, then information about B should not be linked to historical records of A — it means that A and B should have different identifiers.

7. A dynamic application example. In the past we have shown how the global control based on consistent process states can be efficiently applied to control the behaviour of parallel irregular applications [22]. An implementation of a parallel and distributed branch–and–bound algorithm included the observations enumerated below.

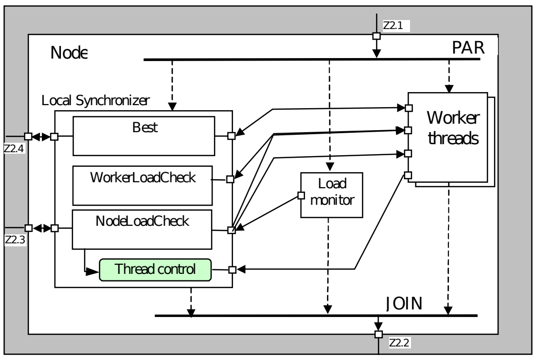


Fig. 7.1: A branch–and–bound search process with dynamic thread creation/deletion

- Each process maintained its local subproblem pool containing partial solutions to be further developed.
- A standard branch–and–bound algorithm was run locally in each process (get a subproblem from a pool, check if it is perspective, branch it, bound each newly created subproblem and return it to the pool).
- The processes reported their load in terms of the estimated size of the subproblems in their pools.
- The synchronizer built global states from those reports to detect load imbalance and dispatched control signals to processes ordering them to communicate and level their loads
- Each new solution better than the currently known best solution was immediately reported to the synchronizer. The synchronizer broadcasted it to all processes using control signals.

PEGASUS let us refine this implementation by introducing thread–level parallelism on each node with a single process running on every node. A local synchronizer on every node can do the same job as the synchronizer for processes in the original solution. To provide the global two–level hierarchical control for all application nodes a global synchronizer which cooperates with the local ones can be introduced [17].

Now, let us assume that the computing system is not used exclusively to solve the branch–and–bound problem. Other tasks can also be run there, possibly with higher priorities. The branch–and–bound application should use all free CPU cores, but should limit itself if other applications are run. Local synchronizers can obtain node load information from a PEGASUS runtime module for CPU load monitoring. It is shown in Fig. 7.1. A single application process (Node) is depicted there. Upon application initialization the local synchronizer, the built–in load monitor and a predefined number of worker threads start executing. The threads report their load and the solutions they find to the synchronizer (to the predicates "WorkerLoadCheck" and "Best", respectively) and they can receive corresponding control signals (broadcasted new best solution and an order to perform load leveling). They can send the "Join" and "Leave" message to the synchronizer as well. The Thread Control subsystem within the synchronizer handles such messages. Finally, the synchronizer reads the node load from the Load monitor (predicate NodeLoadCheck). This predicate can initiate a thread shutdown by sending control signals to a thread: load balancing aimed to transfer the entire load, and a "Stop" signal telling the thread to report "Leave" and then stop. Additionally NodeLoadCheck can trigger the Thread Control module to start a new thread. Visible external ports Z1.4 and Z1.3 are used to connect local synchronizer to the global one.

When the local load exceeds a given threshold, then the local synchronizer may send control signals to some threads ordering them to shut down. The existing load–balancing mechanism can be used to move the remaining load (subproblem pool contents) from the threads signaled to be cancelled to some others. Normally, a thread is told what portion of its load it should transfer. In this case it should just transfer the entire load. The signaled threads will use the "Leave" message to detach from the synchronizer after the complete load transfer is done. Because the detachment is expected by the synchronizer, the problems described in the previous section are void in this case.

On the other hand, if the load is too low, the synchronizer can create a thread to add more computing power for the branch–and–bound search. The new thread will connect to the synchronizer by the "Join" message. Soon, it will be recognized as an underloaded thread and a standard load balancing action will proceed.

Let's note, that changing the computing capacity of nodes due to varying number of running threads is automatically taken into account by the existing load balancing mechanism. When the number of threads diminishes, the subproblem pool is processed slower and it becomes more likely that this node will be considered as an overloaded one. Nodes becoming faster due to an increased number of working threads will process their tasks faster and can be recognized as underloaded nodes.

8. Conclusions. The paper has concentrated on new aspects in designing distributed program execution control based on monitoring of application global states. These aspects concern strongly consistent global states monitoring when the number of processes and threads, whose states contribute to the global application control is changing during programs execution. The proposed solutions are oriented towards the PEGASUS distributed program execution framework, which provides a ready to use infrastructure for designing global control in distributed programs governed by predicates evaluated on global application states. An algorithm for detection of strongly consistent global application states has been proposed. It enhances the theoretical background for the PEGASUS framework and contributes to the domain of the design of evolved control for distributed programs.

The PEGASUS framework is currently under implementation based on many–core processors interconnected by separate networks for control and computational data communication, working under the Linux operating system. As the system implements new parallel program control methods, there are no existing libraries which provide the type of high–level services we need. We use standard parallel libraries to implement the basic functionalities like multi threading or message passing. These libraries are used in a standard way. Only some technical details concerning Linux signals and pthreads interaction proved to be non–trivial and we described that topic in [26].

Inter–process control communication is organized by the use of message passing over Infiniband network. Data communication for computations is executed over an Ethernet network. User program code is written in C/C++ language. MPI2 library is used to express data communication in user programs at the process level. OpenMP and pthreads libraries are used to organize user program execution at thread level with inter–thread communication. The code of the PEGASUS execution framework is written C/C++ language supported by the MPI2, OpenMP and pthreads libraries This paper has been partially sponsored by the MNiSW research grant No. NN 516 367 536.

REFERENCES

[1] N. Carriero, D. Gelernter, *Linda in Context*, in Communications of the ACM 32 (4), April 1989, pp. 444–459

[2] http://www.cs.ucy.ac.cy/courses/EPL441/manifold/tut.pdf, http://reo.project.cwi.nl/

[3] K. Marzullo, D, Wood, *Tools for constructing distributed reactive systems*, Technical report 14853, Cornell University,Department of Computer Science, Feb. 1991,

[4] M. Tudruj, *Fine–Grained Global Control Constructs for Parallel Programming Environments*, in Parallel Programming and Java: WoTUG–20, IOS, 1997, pp. 229–243.

[5] M. Tudruj, J. Borkowski, D. Kopanski, *Graphical Design of Parallel Programs with Control Based on Global Application States Using an Extended P–GRADE System*, in Distributed and Parallel Systems, Cluster and GRID Comp., Kluver, Vol. 777, 2004.

[6] M. Tudruj et al., *Distributed Program Control Flow and Behaviour Governed by Global States Monitoring*, PARELEC 2011, IEEE CS 2011, pp. 73–78

[7] O. Babaoglu et al., *Group Membership and View Synchrony in Partitionable Asynchronous Distributed Systems: Specifications*, in ACM SIGOPS Operating Systems Review, Volume 31 Issue 2, April 1997

[8] A. Bartoli, O. Babaoglu, *Application–based dynamic primary views in asynchronous distributed systems*, in Journal of Parallel and Distributed Computing Volume 63, Issue 4, April 2003, pp. 410–433

[9] L.E. Moser, P.M. Melliar–Smith, D. A. Agarwal, R.K. Budhia, C.A. Lingley–Papadopoulos, T.P. Archambault, *The Totem System*, in Proc. of the 25th Annual International Symposium on Fault–Tolerant Computing, IEEE Computer Society Washington, DC, USA, 1995

[10] N. Lesley, A. Fekete, *Providing view synchrony for group communication services*, Acta Informatica, vol. 40, Number 3, 159–210, DOI: 10.1007/s00236-003-0129- 4, 2003

[11] J. Pereira, L. Rodrigues, R. Oliveira, *Reducing the Cost of Group Communication with Semantic View Synchrony*, in Proc.of the International Conference on Dependable Systems and Networks, 2003

[12] P. Brinch-Hansen, *The Programming Language Concurrent Pascal*, in The Search for Simplicity: Essays in Parallel Programming, Ch. chapter 7, IEEE Computer Society, pp. 58–79, 1996

[13] A.L. Cox et al., *Software versus Hardware Shared–Memory Implementation: A Case Study*, in Proc. Of the 21th Annual International Symposium on Computer Architecture (ISCA'94)

[14] MPI: A Message Passing Interface Standard Ver. 1.1, 1995

[15] M. Tudruj, P. Kacsuk, *Extending Grade Towards Explicit Process Synchronization in Parallel Programs*, Computers and Artificial Intelligence , Vol.17, 1998, pp. 507–516

[16] J. Borkowski, *Global Predicates for On–Line Control of Distributed Applications*, in International Conference on Parallel Processing and Applied Mathematics PPAM 2003, Czestochowa, Poland, Springer LNCS 3019.

[17] J. Borkowski, *Parallel Program Control Based on Hierarchically Detected Consistent Global States*, PARELEC 2004, Dresden, Germany, IEEE, pp 328–333.

[18] O. Babaoglu, K. Marzullo, *Consistent global states of distributed systems: fundamental concepts and mechanisms*, in Distributed Systems (Addison-Wesley, 1995) Consistent global states of distributed systems: Fundamental Concepts and Mechanisms.

[19] Scott D. Stoller, *Detecting Global Predicates in Distributed Systems with Clocks*, Distributed Computing, Volume 13 Issue 2 (2000) pp. 85–98

[20] F. Cristian, F. Schmuck, *Agreeing on processor group membership in timed asynchronous distributed systems*, Technical Report CSE95–428, Department of Computer Science, University of California San Diego

[21] J. Borkowski, *Measuring and improving quality of parallel application monitoring based on global states*, Parallel and Distributed Computing, ISPDC 2005, Lille, France, IEEE.

[22] J. Borkowski, M. Tudruj, *Dual Communication Network in Program Control Based on Global Application State Monitoring*, ISPDC 2007, July, 2007, Hagenberg, Austria, IEEE CS, pp. 37–44.

[23] J. Borkowski, D. Kopanski, M. Tudruj, *Global predicate monitoring applied for control of parallel irregular computations*, Euromicro PDP 2007, February 2007, Naples, Italy. IEEE Computer Society 2007, pp. 105–112.

[24] M. Tudruj, J. Borkowski, L. Masko, A. Smyk, D. Kopanski, E. Laskowski, *Program Design Environment for Multicore Processor Systems with Program Execution Controlled by Global States Monitoring*, ISPDC 2011, IEEE CS, pp. 102–109.

[25] J. Borkowski, M. Tudruj, *Global Control in Distributed Programs with Dynamic Process Membership*, 20th Euromicro International Conference on Parallel, Distributed and Network–based Processing, 2012, pp.525–529.

[26] J. Borkowski, M. Tudruj, A. Smyk, D. Kopanski, *Global Asynchronous Parallel Program Control for Multicore Processors*, Lecture Notes in Computer Science, 2012, Volume 7133/2012, pp. 119–130.

Edited by: Marcin Paprzycki
Received: May 1, 2012
Accepted: June 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_ε 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.