SCALABLE COMPUTING
Practice and Experience

Special Issue: Software Agents Technology

Editors: Shahram Rahimi and Raheel Ahmad

Volume 7, Number 1, March 2006

ISSN 1895-1767

SWPS

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

Scalable Computing: Practice and Experience

Volume 7, Number 1, March 2006

TABLE OF CONTENTS

Scalable Computing: Practice and Experience
Volume 7, Number 1, pp. i–iii. http://www.scpe.org
ISSN 1895-1767
© 2006 SWPS

EDITORIAL: TOWARD A PERFORMANCE EVALUATION METHODOLOGY FOR THE PI-CALCULUS FAMILY OF FORMAL MODELING LANGUAGES

Process calculi are tools capable of modeling large distributed systems of concurrent processes and among them Pi-calculus, introduced by Milner et al. in 1992 (Milner 1992), is one of the most popular ones. Pi-calculus can be used to model modern concurrent systems ranging from the lowest level in software design, procedural programming, to the highest level, component composition and abstraction (Milner 1999 and Sangiorgi 2001). The Pi-calculus has aroused intensive interests in researchers to study its applicability, extensibility, and other properties. For example, it has been shown that Pi-calculus is powerful enough to model various data structures, object-oriented programs, and communication systems. It is also the basis of several experimental programming languages such as Pict (Pierce 2000), Join (Fournet 1997), and TyCO (Vasconcelos 1998), etc. The Pi-calculus is a calculus where both communication and configuration are primitives (Milner 1992) and has been successfully used to model features of object-oriented programming languages (Walker 1995).

My experience supports the fact that the Pi-calculus is a promising formal foundation for modeling concurrent objects (Rahimi 2003 and Shaw 1996) and for the definition of higher-level, reusable synchronization abstractions (Schneider 1997). Other research initiatives have proven that the Pi-calculus can be used to effectively and precisely specify the configuration and behavior of concurrent programs and to deduce their behavioral properties (Radestock 1994). With all the advantages that Pi-calculus offers, it does not provide any formal method to be used for performance evaluation of systems it describes. In this editorial, I present a direction for the design and implementation of a performance evaluation module by using the Markov Chain and Markov Reward Model (Bolch 1998).

To be able to construct a Markov Chain for a formally described system, we plan to expand the Pi-calculus by adding some performance primitives and associating performance parameters with each action that takes place internally in a system. These include actions such as sending and receiving messages from one component to another component. By using such parameters, designers can benchmark their component-oriented units and compare the performance of different architectures against one another.

Markov Chain (MC) was introduced by A. A. Markov in 1907 and has been in use for performance analysis since 1950. A Markov chain consists of a set of states and a set of labeled transitions between these states and is a sequence of random values whose probabilities at a time interval depend upon the values at the previous states. There exist two types of Markov Chains: Continuous-Time Markov Chains (CTMCs) and Discrete-Time Markov Chains (DTMCs). CTMCs are distinct from DTMCs in the sense that state transitions may occur at anytime and not merely at fixed, discrete time points, as is the case with DTMCs. More information about Markov Chains can be found in (Bolch 1998).

By proving that a system modeled by Pi-calculus (or an extension of it) is CTMC, the performance evaluation methodology could be built using Markov Chains theory. The steps needed to be taken for MC-based performance evaluation of a formal model described in Pi-calculus or its extensions are projected to be as follow:

1. Forming the state diagram of the system (based on MC theory): To do so, we consider the original system expression in Pi-calculus to represent the initial state of the system. Every reduction applied to the initial system expression would produce a possible next state.
2. Utilizing the state transition rate functions to calculate the rate for each transition: The transition rate function is defined as the total number of the transitions in the unit time. From (Bolch 1998), we know that the state sojourn times of a homogeneous CTMC must have the memory less property (does not remember the previous states). Since the exponential distribution is the only continuous distribution with this property, the random variables denoting the sojourn times (t_i), also called holding times, must be exponentially distributed. If r_i denotes the i^{th} transition rate, it obviously follows that $r_i = 1/t_i$. Therefore, for the purpose of this development, t_i is based on the exponential distribution with the parameter (λ) equal to r_i.
3. By associating each rate with its appropriate transition in the state diagram, the Markov Chain diagram would be formed.
4. If it is an ergodic Markov Chain, defined in (Bolch 1998), the steady probability of each state (π) could be found by solving the equation: $\pi * Q = 0$, where Q is the infinitesimal generator matrix, defined in

(Bolch 1998). The final performance value of the system would be equal to: $\sum_{i=1}^{n} r_i * \pi_i$, where r_i and π_I are the reward and the steady probability for the state S_i.

5. If it is not an ergodic Markov Chain, which means the system will eventually stop at some certain states, then the reward will be assigned for each transition separately. In this case, we can only calculate the performance of the system executing from one state to another.

By utilizing Markov state-based and transition-based reward models, performance evaluation can be executed for systems described in Pi-calculus or its extensions. The performance evaluation module could be integrated into a visualization tool, such as ACVisualizer (Rahimi 2006), so the user, after visualizing component composition, can verify the model and evaluate its performance against other architectures before finalizing the design for implementation.

Here is a basic example to illustrate the above steps. In this example, a system is composed from two main components: a Memory and a Processor. The memory unit waits on the channel to receive an address and then uses another channel to send the data out. The processor sends an address to the memory, waits for the data to be received and then continues executing the instruction (τ):

$$Mem \equiv req(addr).\overline{ans}\langle data\rangle.Mem$$
$$\mathrm{Pr}\,ocessor \equiv \overline{req}\langle addr\rangle.ans(data).\tau.\mathrm{Pr}\,ocessor$$

According to the first step, described above, we first have to reduce the model step by step using Pi-calculus reduction rules. Each reduction step introduces a state which can be used to form the state diagram in figure 1a. In this figure:

$$S_1 \equiv req(addr).\overline{ans}\langle data\rangle.Mem \mid \overline{req}\langle addr\rangle.ans(data).\tau.\mathrm{Pr}\,ocessor$$
$$S_2 \equiv \overline{ans}\langle data\rangle.Mem \mid ans(data).\tau.\mathrm{Pr}\,ocessor$$
$$S_3 \equiv Mem \mid \tau.\mathrm{Pr}\,ocessor$$

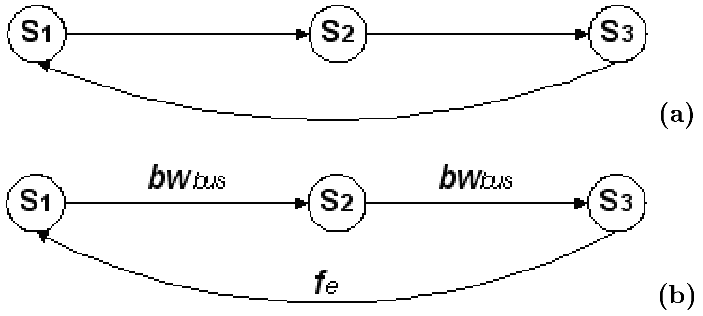


FIG. 1. *a) the State Diagram b) an ergodic Markov Chain*

The speed of sending and receiving data between the processor and the memory depends on the bandwidth of the bus (noted as bw_{bus}). So we use $1/bw_{bus}$ to denote the communication cost between the processor and the memory (based on step 2). The cost of executing the action τ is denoted as $1/f_e$ where f_e is the clock frequency of the chipset. Based on this information, the state diagram can be transferred to the ergodic Markov Chain in figure 1b (step 3).

From step 4, suppose the steady probability vector is $\pi = (\pi_1, \pi_2, \pi_2)$. From figure 1b, we know the infinitesimal generator matrix is:

$$Q = \begin{pmatrix} -bw_{bus} & bw_{bus} & 0 \\ 0 & -bw_{bus} & bw_{bus} \\ f_e & 0 & -f_e \end{pmatrix}$$

Since $\pi * Q = 0$ and $\pi_1 + \pi_2 + \pi_3 = 1$, we can calculate π:

$$\pi = \big(f_e/(bw_{bus} + 2f_e), f_e/(bw_{bus} + 2f_e), bw_{bus}/(bw_{bus} + 2f_e)\big).$$

Suppose we want to evaluate the throughput of the system, then the rewards for states S_1, S_2, and S_3 are 0, 0, 1. The final performance equation, based on step 4 is:

$$0 * f_e/(bw_{bus} + 2f_e + 0 * f_e/(bw_{bus} + 2f_e) + 1 * bw_{bus}/(bw_{bus} + 2f_e) = bw_{bus}/(bw_{bus} + 2f_e),$$

which, if given the numerical values for bw_{bus} and f_e, can provide the calculated throughput of the system.

In summary, with all the advantages that utilization of a formal modeling tool may offer to the design and implementation of a sound and solid distributed system, the lack of a formal infrastructure for performance evaluation compromise its practical utilization. Although the Pi-calculus family of formal languages is vastly utilized, it does not provide any native theoretical mechanisms for performance evaluation of different design approaches for a particular system. This deficiency makes Pi-calculus incapable in assisting the user with design-for-performance which is a crucial component of practical application development. This editorial plots a path to address this need in Pi-calculus.

Shahram Rahimi,
Department of Computer Science
Southern Illinois University.

REFERENCES

[1] G. Bolch, S. Greiner, H. Meer, and K. S. Trivedi, *Queuing Networks and Markov Chains Modelling and Performance Evaluation with Computer Science Applications,* Wiley, New York. Borland International (1995), Delphi Benutzerhandbuch, 1998.

[2] C. Fournet and L. Maranget, *The Join-Calculus Language (release 1.05),* Institut National de Recherche en Informatique et Automatique, `http://pauillac.inria.fr/join/manual/index.html` 1997.

[3] R. Milner, J. Parrow and D. Walker, *A calculus of mobile processes (Parts I and II),* Information and Computation, vol. 100, no. 1, pp. 1–77, 1992.

[4] R. Milner, *Communication and Mobile Systems: The π-Calculus,* Cambridge University Press, Cambridge, UK, 1999.

[5] B. Pierce and D. Turner, *Pict: A programming language based on the Pi-calculus,* In Proof, Language and Interaction, MIT Press, 2000.

[6] S. Rahimi, *Using Api-Calculus for Formal Modeling of SDIAgent, a Multi-Agent Distributed Geospatial Data Integration System,* Journal of Geographic Information and Decision Analysis, ISSN 1480-8943, vol. 7, No. 2, pp. 132–149, 2003.

[7] S. Rahimi and R. Ahmad *ACVisualizer: A Visualization Tool for API-Calculus,* to appear in Multi-Agent and Grid Systems, 2006.

[8] M. Radestock and S. Eisenbach, *What do you get from a pi-calculus semantics?* Proceedings of Parallel Architectures and Languages Europe (PARLE '94), LNCS 817, pp. 635–647, Springer, 1994.

[9] D. Sangiorgi and D. Walker, *The π-calculus: A Theory of Mobile Processes,* Cambridge University Press, Cambridge UK, 2001.

[10] J. Schneider and M. Lumpe, *Synchronizing Concurrent Objects in the Pi-Calculus,* Proceedings of Languages and Models with Objects '97, Brest, 1997.

[11] M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline,* Prentice Hall, 1996.

[12] V. Vasconcelos and R. Bastos, *The TyCO Programming Language,* `http://www.ncc.up.pt/ lblopes/tyco/` 1998.

[13] D. Walker, *Objects in the Pi-Calculus,* Information and Computation, vol. 116, no. 2, pp. 253–271, 1995.

Scalable Computing: Practice and Experience
Volume 7, Number 1, p. v. http://www.scpe.org

ISSN 1895-1767
© 2006 SWPS

GUEST EDITOR'S INTRODUCTION: SOFTWARE AGENT TECHNOLOGY

Software agent technology is an exciting paradigm which can be efficiently applied to many distributed computing problems, particularly those that require dynamic behavior to reach a solution. Software agent technology has emerged as an enhancement of, if not an alternative to, the traditional client/server model in such an environment. Mobile agents can migrate to a desired remote peer and take advantage of local processing rather then relying on remote procedure calls (RPC) across the network, as in the case of client/server systems. Also, agents are entities which function continuously and autonomously in a particular environment, and are able to carry out activities in a flexible and intelligent manner that is responsive to the dynamically changing environment. Ideally, an agent that functions continuously would be able to learn from experience and have capabilities to adapt to ad-hoc events and apply suitable functionality depending on the circumstances. Employment of the agent technology in the traditional client-server environments could result in more robust, flexible and better performing solutions for many applications.

In this special issue, we have collected five cutting edge research papers which are the extended versions of a selected group of the papers already published in "Intelligent Agent Systems 2005 (IAS 05) special track" at the 18th International FLAIRS Conference. The first paper, "Agents Go Traveling", by D. O'Kane, D. Marsh, S. Shen, R. Tynan and G. M. P. O'Hare, is concerned with the infrastructure support for nomadic agents. The authors have introduced the Agent Travel Metaphor (ATM) which offers a metaphor fostering integration of control and security. The second paper is entitled "Exploiting Shared Ontologies with Default Information for Web Agents", by Y. Ma, B. Jin, and M. Zhou. This paper uses distributed description logic (DDL) to model the mappings between ontologies used by different agents and further makes an extension to the DDL model. "A WS-Agreement Based Resource Negotiation Framework for Mobile Agents" is the title of the third paper by D. G. A. Mobach, B. J. Overeinder, and F. M. T. Brazier. This paper presents a negotiation infrastructure with which agents acquire time-limited resource contracts through negotiation with one or more mediators instead of individual hosting systems. The fourth paper, "Agent Composition via Role-Based Infrastructures", by G. Cabri, proposes to build infrastructures based on roles, which are abstractions that enable the composition of different agents in an open scenario. Finally the last paper, unlike the first four papers, presents an application of software agents for support of student mobility. This paper is authored by M. Ganzha, W. Kuranowski and M. Paprzycki.

We have tried our best to present quality works in this special issue and certainly hope that we have achieved this goal.

Shahram Rahimi,
Raheel Ahmad
Department of Computer Science
Southern Illinois University
Mailcode 4511, Carbondale
Illinois 62901-4511

Scalable Computing: Practice and Experience
Volume 7, Number 1, pp. 1–10. http://www.scpe.org

ISSN 1895-1767
© 2006 SWPS

AGENTS GO TRAVELING

DONAL O'KANE , G. M. P. O'HARE*, DAVID MARSH , SONG SHEN , AND RICHARD TYNAN

Abstract.
This paper is concerned with infrastructural support for nomadic agents. Agent migration provides a wide range of advantages and benefits to system designers, however issues relating to security and integrity mobile agents has mitigated against the harvesting of their true potential. Within this paper we introduce the Agent Travel Metaphor (ATM) which offers a comprehensive metaphor fostering integrating of control and security for mobile agents. We describe the metaphor together with its incorporation within the Agent Factory multi-agent system.

1. Introduction. In recent years much attention has been focused on the area of multi-agent systems and mobile agents. The term agent has numerous connotations and can represent various software entities.

In this paper the term agent implies an entity comprising of but not limited to the following properties: autonomy, social ability, responsiveness, pro-activeness, adaptability, mobility, veracity, rationality, and human cognition modeling techniques such as belief desire and intention. These define a strong notion of agency, [18]. A mobile agent refers to the capacity of an agent to electronically navigate a network in which it exists, [17].

Many arguments have been proffered as to the benefits and disadvantages of mobile agents, [5], and on the use of agent-oriented programming as a design paradigm, [11].

Numerous pitfalls have been identified, such as potential security issues involved in agent migration, [10, 5, 14, 4], interoperability and language translation, [13, 8, 2] and dynamic creation and management of an agent's itinerary, [15].

Various solutions to these drawbacks have been offered by which to address the difficulties associated with agent migration. If solutions could be found to these impediments, then the potential benefits to be harvested from secure agent mobility are immense. As yet, no system or approach has garnered universal support. The itinerant agent framework, [4] is an example of a framework and design that, allows agent platforms to offer a selection of services targeted at roaming agents, with no specific origin or home. [16] introduce the concept of airports for internet agents. These airports provide a framework for ad-hoc and unstable internet agents to access resources and maintain communicate channels. However these are targeted at specify types of mobile agents, not at the general population.

To address this lack of unity this paper introduces the Agent Travel Metaphor, (ATM), which describes and provides a natural method to deploy and implement a vast range of tools and services designed to offer mobile agents various services.

The metaphor draws from the experience of human travel and utilizes techniques from the natural world to provide services such as security, adaptation, core code protection, cooperation and interoperation.

This paper also proposes that the use of the ATM may provide a service backbone for mobile agents to exist, operate and lengthen their lifespan, by providing a range of services and adaptation mechanisms.

2. Mobile Agents—The Way Forward.

2.1. Related Research.

The ability of an agent to migrate is defined as electronic transfer of an agent from one point in a computer network to another, [17]. Many multi-agent systems support agent mobility and provide services to enable their agents to migrate efficiently.

A diversity of enabling technologies have been adopted in order to underpin agent mobility. These include mobility security measures, agent language translation, middle agents, load balancing mechanisms, ad-hoc network tools and agent code protection.

Mobility Security: A variety of security issues arise when enabling agents with mobility. The main issues are protection of platforms from malicious agents, protection of agents from malicious platforms, protection of agents from malicious agents and protection of agents and platforms against other entities [10]. Many systems have been developed to tackle these four problems, including design techniques that limit an agents ability to affect the agent platform and scanning systems that ensure migrating agent are not carrying harmful code such as viruses and systems that utilize passport and visa documents and digital certificates, [9, 4], to enable

*Adaptive Information Cluster (AIC), Department of Computer Science, University College Dublin (NUI), Belfield, Dublin 4, Ireland. Gregory.OHare@ucd.ie

identification and the origin of migrating agents, and thus provide agent platforms with a method to measure how trustworthy a migrating agent is.

Agent language translation: If an agent is migrating between heterogeneous multi-agent systems then it is possible for an agent to migrate to an agent platform that utilizes a different agent communication language, (ACL), than the migrating agent. Agent language translation allows the migrating agent to communicate and cooperate with foreign agent platforms as well as foreign agents. Services and agents that facilitate interoperation between heterogenous multi-agent systems exist and use meta languages, [2], direct translation, [4], or ontologies,[8], to provide agents with a means of language translation.

Middle agents: These are agents that provide a variety of static services to mobile agents. They are generally accepted to be system level agents or trusted agents that provide services such as naming and access to shard resources, [7, 4]. Systems such as the the itinerant agents framework, [4], and the SOMA system, [1], use a variety of middleware software to ensure security, interoperability and communication services to mobile agents.

Load balancing mechanisms: Tools and services that provide load balancing for their network have been developed, [3].

Ad-hoc network tools: The ad-hoc nature of ever changing networks presents another massive set of problems for a mobile agent environment. For example, node disappearance and sudden unannounced reappearance causes a serious failure and recovery discovery problem. Nodes that fail are no longer reachable, agents executing on such nodes are also no longer reachable. Any dependant agents or services must either have the ability to handle this failure or be notified of its occurrence. Upon recovery or re-establishment of a node connection, the other nodes or entities must be made aware of the recovery, this is a non-trivial problem, [16].

Agent code protection: Protection for the core of an agents mental state against viral code and unauthorized access has been developed, [12]. Agent platform administrators can assume benevolence of known agents once their critical code is protected.

2.2. The Agent Travel Metaphor. The Agent Travel Metaphor, (ATM) adopted and introduced within this paper, borrows heavily from human travel. It consolidates and expands previous work by other researchers that have adopted components and segments of the overall travel scenario. Three classes of human travel can be identified as useful for mobile agents, *International* travel, *National* travel and *Metropolitan* travel. In the human environment these classes of travel all possess diverse procedures that determine issues such as how travel is initiated, how secure the travel is and a plethora of further services provided for the travelers.

Metropolitan: An example of metropolitan travel would be traveling on a bus through a city. For a traveler that lives in that city this form of travel will result in destinations that are both similar to the origin location and familiar to the traveler. The origin and destination locations are almost identical, travel can be initiated at any time with little or no preparation, security measures at at a minimum, the traveler has no requirement to prove their identity and must only purchase a ticket for a bus at the time of travel, and travelers already possess most of what they need to carry out their tasks at the destination, (language translation, behavior learning etc).

National: An example of national travel would be traveling on a train from one city to another, within the same country. This method of traveling is much stricter than metropolitan travel, there are more stringent security measures, the travelers must usually purchase a ticket in advance and must present this ticket upon request. The schedule is also much more regulated, train time tables are usually much more strictly adhered to than a metropolitan bus timetable, also a traveler may be required to pack items necessary for their tasks as they may not be available at the destination.

International: An example of international travel would be traveling on an airplane from one country to another. This method of travel has the highest level of security requirements, passport, visa, ticket and baggage checks at both origin and destination, a extremely limited timetable, only a few flights per day to particular destinations, and a limited choice of direct destinations, requiring several stops to get to a particular destination. Along with these is the possibility for a traveler to arrive at a destination with a different language and different behavior as standard, the traveler must be taught or discover how to conform to these new requirements. Packing baggage is most important for international travelers as it becomes more difficult for them to procure the necessary items at foreign destination that will enable them to perform their tasks.

An example of the steps taken by a traveler undertaking an international class of travel involves the following steps:

- Decide on destination(s).
- Contact a travel agent and negotiate a ticket price.
- Travel agent issues a ticket.
- Traveler contacts destination for visa requirements.
- Contact vaccination clinic.
- Query and obtain necessary vaccinations for destination(s).
- Pack necessary baggage.
- Upon arrival at the origin port contact port authorities.
- Ticket and passport are verified and baggage is scanned/checked in.
- Port initiates migration.
- Upon arrival at destination contact destination port authorities.
- Ticket and visa are verified and baggage is scanned and collected or left in secure box.
- Once through security measures contact language translator and/or behavior teacher as needed.
- Normal operation resumes.

```
Begin
    Select Destination                            // (a)
    Get Destination Description                   // (b)
    if( ! same language ){
        Contract Translator                       // (c)
    }
    if( ! same behavior ){
        Contract Teacher                          // (d)
    }
    Determine Packing Requirements                // (e)
    if( packing necessary){
        Pack Baggage                              // (f)
    }
    Obtain Necessary Travel Documents             // (g)
    Submit Travel document to Port Authority      // (h)
    Migrate once Authorized                       // (i)
End
```

Fig. 2.1. *The agent travel algorithm.*

In the metaphor, agents play the role of travelers, an evolution from the human travel analogy, as agents are typified by their use of human cognition techniques in their decision making process.

Accordingly, agent platforms play the role of cities, also a natural evolution from the travel analogy as an agent platform is the location in which an agent exists and interacts with its surrounding environment. The potential for agent platforms to create federations with groups of agent platforms and the analogous relationships between neighboring cities and countries, e.g. countries that have travel agreements and do not require visitors to apply for a visa to enter. This reinforces the previous argument in favor of the metaphor.

The variety of processes that we go through when we undertake a journey is unique for every journey and yet distinct patterns can be extracted, for example generalizing the security requirements, documents needed for travel and inquiring at the destination if language translation or behavior teaching is required leads to a generalized travel process as seen in Figure 2.1. The ATM is designed to facilitate configuration and policing of mobile agents policies and services in a heterogenous environment. The algorithm described in Figure 2.1 proposes a foundation for a framework, providing agent platforms, and their agents, with modular, secure, agile and adaptive agent mobility services. The provision of these services, allows for a flexible and easily configured environment, giving platforms the capability to create and control affiliations with other platforms and multi-agent networks.

2.3. Important Actors and Data Structures within the Agent Travel Metaphor. In order to implement an initial realization of the agent travel metaphor and its accompanying framework for incorporation within Agent Factory, it is first necessary to define some of the principle actors and data structures necessary to support the metaphor.

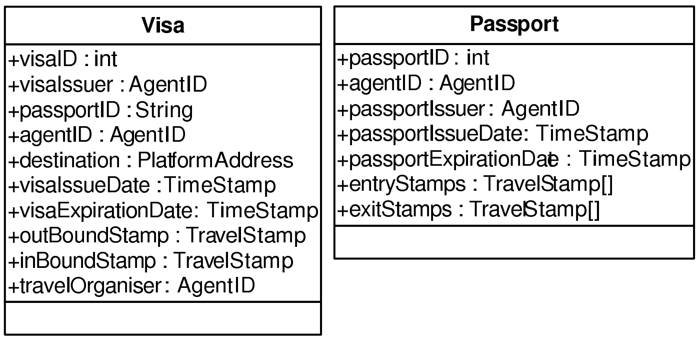

Visa
+visaID : int
+visaIssuer : AgentID
+passportID : String
+agentID : AgentID
+destination : PlatformAddress
+visaIssueDate : TimeStamp
+visaExpirationDate: TimeStamp
+outBoundStamp : TravelStamp
+inBoundStamp : TravelStamp
+travelOrganiser: AgentID
Passport
+passportID : int
+agentID : AgentID
+passportIssuer: AgentID
+passportIssueDate: TimeStamp
+passportExpirationDate : TimeStamp
+entryStamps : TravelStamp[]
+exitStamps : TravelStamp[]

Fig. 2.2. *The Visa and Passport Data Structures.*

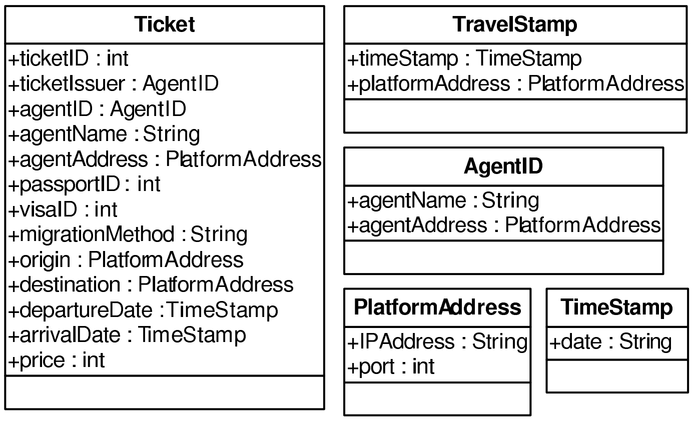


Fig. 2.3. *The Ticket, TravelStamp, AgentID, PlatformAddress and TimeStamp Data Structures.*

Passport Passports are an official certificate issued by a trusted source providing the identity of an agent and providing information on, an agent's origin, the creator of the passport and a history of an agent's travels.

Passport Issuer A passport issuer is a middle Agent contracted to create passport data structures for agents. Passport Issuers retain a copy of all created passports and provide a verification service for any agent wishing to ensure a particular passport is genuine.

Ticket Tickets are certificates that prove that a ticket holder has procured permission for a scheduled migration event.

Travel Organizer A travel organizer middle agent that is contacted by a traveling agent wishing to obtain permission to travel to a particular destination. The travel organizer contacts destinations and procures visas and creates a ticket for the traveling agent. Travel organizers also provide a verification service for any agent wishing to ensure a particular ticket is genuine.

Visa Visas are temporary, once off, certificates that are provided by destination platforms, providing an agent with entry permission to a destination platform.

Port a conceptual location at which all migration to and from an agent platform is coordinated.

Port Authorities Port Authorities are trusted agents charged with the task of operating the port for each agent platform. Each agent platform contains both a port and a port authority agent. The responsibilities of this agent include coordinating agent migration, upholding the local security policies and validating tickets, visas and passports.

Air Side a restricted area of an agent platform. Once An agent commences migration it is restricted from normal operation until it arrives landside at its destination.

Land Side the term used for the normal space for agent operation.

Baggage a collection of code or data, external to an agent's mental state, that the agent makes use of in order to fulfil its goals.

Secure Box a secure and private storage location attached to a port. An agent carrying baggage may deem portions unnecessary for the current location. These unnecessary portions can be stored ready to be retrieved once the agent requires them or leaves the current location.

Vaccinations a security and protection measure that allows agents to defend themselves from infections before migrating to a potentially dangerous/malicious location as well as allowing agent platforms to guard themselves from unknown migrating agents.

Language Translator an agent that can be contracted by a mobile agent to bestow the ability to converse with other agents that use different communication languages.

Behavior Teacher an agent that can be contracted to give an agent the ability to adapt to local operating behaviors. Some platforms within the network may require agents to register with its white/yellow pages services for example, while other locations may not.

3. Agent Factory and the Agent Travel Metaphor.

3.1. Agent Factory Mobility Support. Agent Factory, [6], is a multi-agent systems developed using the strong notion of agency. Agent Factory provides support and infrastructures that allow for rapid prototyping of agents. It imbues its agents with mobility via HTTP socket connections, transferring agent mental state and serialized java code. Federations of agent management services, (AMS), and directory facilitators, (DF), provide white and yellow pages services that supply agent and service naming.

3.2. Enabling Agent Factory with the ATM. In order to evaluate the usefulness of the agent travel metaphor, we identified and extracted a subset of this architecture to be initially implemented. This subset consists of the operations `(a)`, `(g)`, `(h)` and `(i)` defined in Figure 2.1. These operations give rise to the creation of three middle agents and three key data structures.

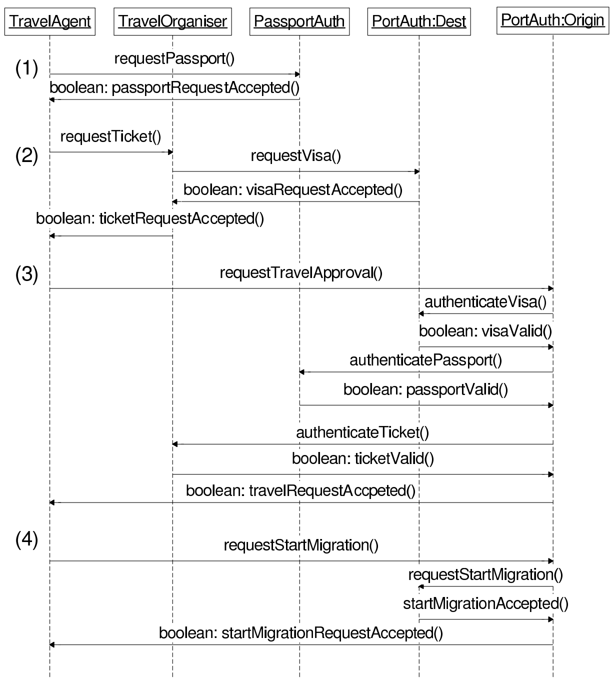


FIG. 3.1. *UML interaction diagram show the 5 implemented agents from the agent travel metaphor and the sequence of messages that occur when an agent migrates using the passport, visa and ticket system of authentication.*

The middle agents, TravelOrganiser, PassportAuthority and PortAuthority are responsible for issuing the three key data structures, Tickets, Passports and Visas, Figure 2.3 and Figure 2.2. These agents also have the responsibility for issuing appropriate travel documents to traveling agents upon request and proper authorization as in Figure 3.1 sections (1) and (2). The TravelOrganiser, PassportAuthority and PortAuthority also must provide a verification service allowing other middle agents that are inspecting travel documents to request that the creator of travel certificates verify that said certificates are valid, Figure 3.1 section (3). In conjunction with providing a verification service for Visa documents, PortAuthority agents are responsible for initiating authentication of incoming and outgoing agents' travel documents, Figure 3.1 section (4).

3.3. Modular Deployment of the Agent Travel Metaphor. The use of middle agents to imbue an agent platform with the ATM takes advantage of the flexibility inherent in intelligent agents, giving rise to the modular nature of the metaphor.

The middle agents, for example the PortAuthority agent, can use locally written software to perform their tasks. This means that the *exact* protocols that the agents use to enforce their security policies can be defined by a local developer or administrator.

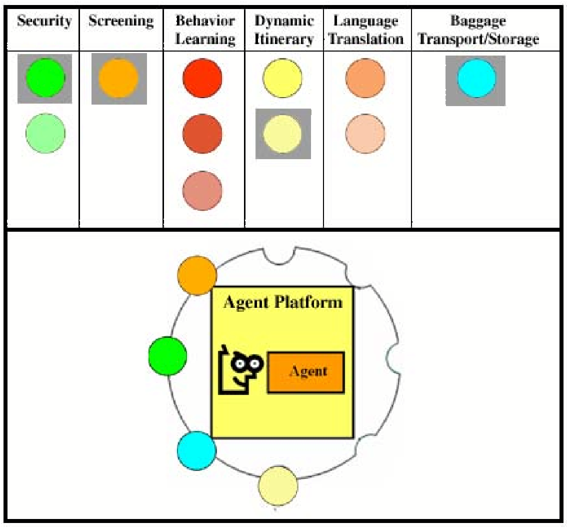


Fig. 3.2. *An agent platform configured with security, screening, dynamic itinerary, and baggage trasport/storage services.*

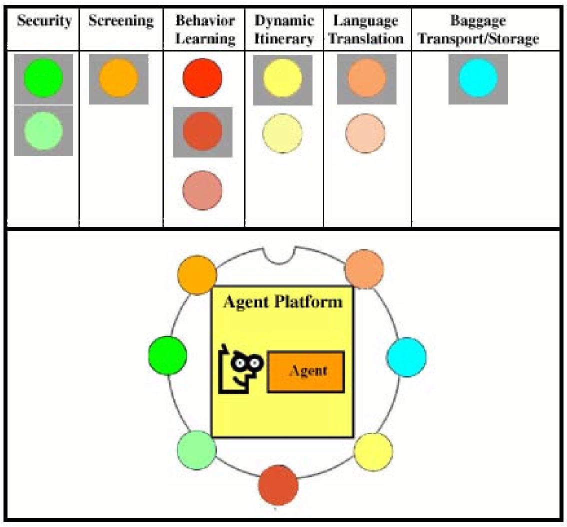


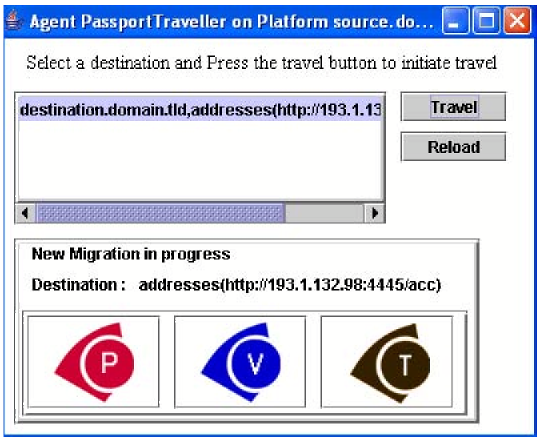
Fig. 3.3. *An agent platform re-configured with several extra services, behavior learning and language translation.*

Platform administrators can use the modular structure of the ATM to setup policies for services such as security, language translation etc, Figure 3.2. Administrators can also dynamically modify the services that exist on a particular platform as well as edit and augment existing services on a platform, Figure 3.3.

Consider the following scenario: An administrator controls a particular set of five agent platforms. The administrator knows that agent language and behavior are identical across all of the platforms and that only one of the platforms has access to critical assets that need to be protected.

The administrator instructs the critical platform's PortAuthority agent to demand that incoming migrating agents present Ticket, Passport and Visa documents. Furthermore it must scrutinize presented travel documents aggressively, verifying them with their issuers along with performing a query to federated PortAuthority agents to ensure that the migrating agent or its origin has not been blacklisted or quarantined for misbehavior.

An agent, PassportTraveller, decides that it wishes to migrate to the critical platform in order to access the secure data there. The agent is informed of the requirements set by the administrator, a passport, visa and ticket, by the TravelOrganiser agent. The agent then either contacts the appropriate agents in order to satisfy these requirements if it has knowledge of the behavior that is necessary to do so or, if the agent does not currently have the necessary knowledge to satisfy the migration requirements the agent can contact a behavior teacher to learn how to satisfy the requirements so that it may migrate.



Select a destination and Press the travel button to initiate travel

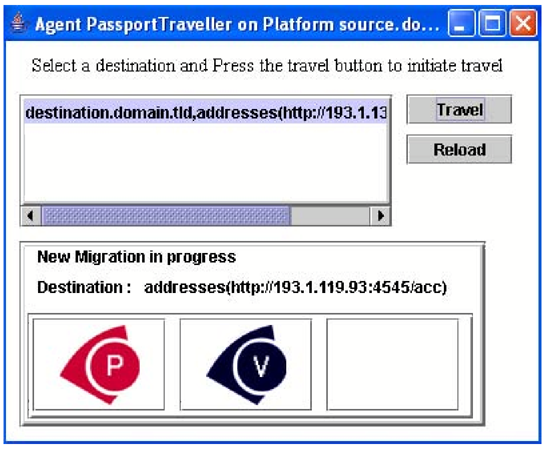
destination.domain.tld,addresses(http://193.1.13

New Migration in progress

Destination : addresses(http://193.1.132.98:4445/acc)

Fig. 3.4. *A screen shot showing an agent, PassportTraveller, migrating to destination* `http://193.1.132.98:4445`

Figure 3.4 shows an agent, PassportTraveller, that is migrating to the critical platform. The agent, as required, must satisfy three requirements, presenting a passport, visa and ticket in order to successful migrate. Figure 3.4 shows the agent has successfully obtained the first two travel documents as the red passport and blue visa icons are no longer grey, and is waiting upon the ticket document so that it can proceed with its migration.



Select a destination and Press the travel button to initiate travel

destination.domain.tld,addresses(http://193.1.13

New Migration in progress

Destination : addresses(http://193.1.119.93:4545/acc)

Fig. 3.5. *A screen shot showing an agent, PassportTraveller, migrating to destination* `http://193.1.119.93:4545`

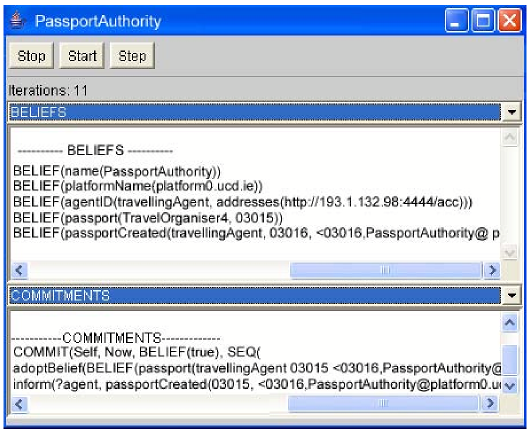
The administrator can set much looser security policies on the other platforms, as security threats are not as potentially catastrophic and damaging to these platforms. The administrator instructs these PortAuthorities to require a Passport and Ticket from incoming migrating agents and to assume agent benevolence, i. e. to accept all presented documents as valid without verifying the documents with their issuer.

Figure 3.5 shows the agent PassportTraveller migrating to another platform with fewer security policies in place. Here the agent is only required to satisfy two requirements, presenting a passport, visa in order to successfully migrate. A ticket is unnecessary as the cost of the migration and the frequency and scheduling of migration between these agent platforms is of minimal importance. This agent has previously satisfied the passport requirements and is awaiting delivery of a visa document before it can proceed with its migration.

The above scenario describes the manner in which agent platforms can be configured in different manners using the ATM and the concepts of *National*, *International* travel and *Metropolitan* travel outlined in the Agent Travel Metaphor section above.

4. Evaluation and Results. In order to evaluate the consequences of enabling Agent Factory with the ATM framework we must consider several issues.

Security: The ATM framework puts into place a configurable set of security measures that allow administrators to set security policies in the manner that they see fit. Figure 4.1 and Figure 4.2 show a PassportAuthority agent and a PortAuthority agent respectively, these agents have been configured to provide a high level of security. In this example the PortAuthority agent is requiring that the traveling agent presents three travel documents, a passport, a visa and a ticket. Along with requiring traveling agents to present these documents the PortAuthority agent also verifies all of the documents' authenticity. These agents can easily be reconfigured to provide a slackened security policy for example if the PortAuthority only required traveling agents to present a single travel document, a passport, and if the PortAuthority agent did not verify the documents authenticity.



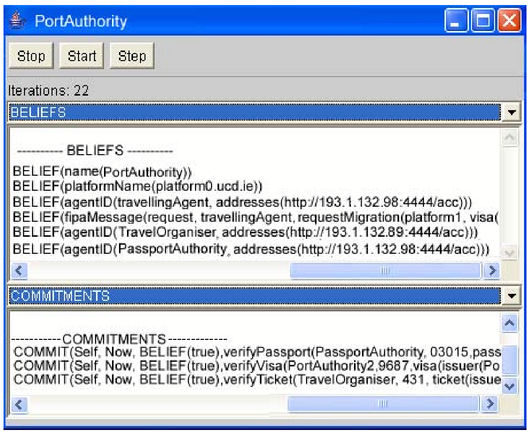
```
Iterations: 11

---------- BELIEFS ----------
BELIEF(name(PassportAuthority))
BELIEF(platformName(platform0.ucd.ie))
BELIEF(agentID(travellingAgent, addresses(http://193.1.132.98:4444/acc)))
BELIEF(passport(TravelOrganiser4, 03015))
BELIEF(passportCreated(travellingAgent, 03016, <03016,PassportAuthority@ p

-----------COMMITMENTS------------
COMMIT(Self, Now, BELIEF(true), SEQ(
adoptBelief(BELIEF(passport(travellingAgent 03015 <03016,PassportAuthority@
inform(?agent, passportCreated(03015, <03016,PassportAuthority@platform0.u
```

FIG. 4.1. *A PortAuthority agent receiving a migration request from a migrating agent, TravellingAgent, the PortAuthority agent handles this request by firstly verifying the documents legitimacy with the issuers.*

Dynamic itinerary: The TravelOrganiser agent provides mobile agents with a method to choose their migration destination without any previous knowledge the location of this platform. This allows agents to create a random migration pattern incorporating new additions to the agent platform network.

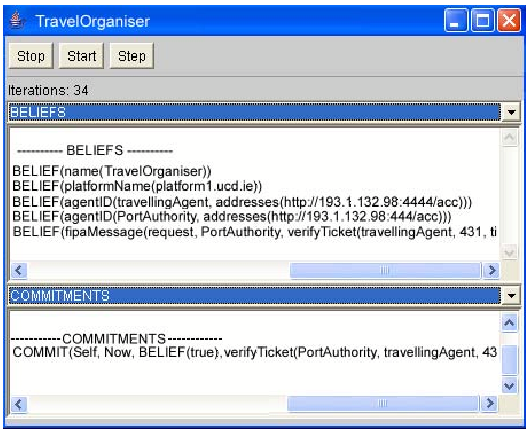
Scheduled migration: When an agent purchases a Ticket, the destination agent platform is informed that an agent wishes to migrate by the Ticket issuer. The origin and destination PortAuthority agents can prepare for the migration event and utilize the time beforehand to modifying the migration schedules on the network.

Migration time: The actual time taken to electronically migrate an agent increases from 5% to 7%, as a result of the agent keeping a record of its travel documents. Total time taken, from the initial decision to initiate a migration until the resumption of operation at the destination is substantially increased by 50% to potentially greater than 300%. The percentage increase can be apportioned to the potential for a large increase in the number of agents that are involved in any migration event. In the examples described above and seen in Figure 3.1, five agents are involved in the migration process, the traveling agent, the PassportAuthority agent, the TravelOrgainiser agent and the two PortAuthority agents, (based at the origin and destination). The number of messages that are passed between these five agents increases from 6 without any of the security measures from the ATM to 19 in the outlined example for the agent to acquire a passport, visa and ticket and to present these documents to the port authorities, and for the port authorities to verify the documents with



```
Iterations: 22

---------- BELIEFS ----------
BELIEF(name(PortAuthority))
BELIEF(platformName(platform0.ucd.ie))
BELIEF(agentID(travellingAgent, addresses(http://193.1.132.98:4444/acc)))
BELIEF(fipaMessage(request, travellingAgent, requestMigration(platform1, visa(
BELIEF(agentID(TravelOrganiser, addresses(http://193.1.132.89:4444/acc)))
BELIEF(agentID(PassportAuthority, addresses(http://193.1.132.98:4444/acc)))

-----------COMMITMENTS------------
COMMIT(Self, Now, BELIEF(true),verifyPassport(PassportAuthority, 03015,pass
COMMIT(Self, Now, BELIEF(true),verifyVisa(PortAuthority2,9687,visa(issuer(Po
COMMIT(Self, Now, BELIEF(true),verifyTicket(TravelOrganiser, 431, ticket(issue
```

Fig. 4.2. *A PortAuthority agent receiving a migration request from a migrating agent, TravellingAgent, the PortAuthority agent handles this request by firstly verifying the documents legitimacy with the issuers.*



```
Iterations: 34

---------- BELIEFS ----------
BELIEF(name(TravelOrganiser))
BELIEF(platformName(platform1.ucd.ie))
BELIEF(agentID(travellingAgent, addresses(http://193.1.132.98:4444/acc)))
BELIEF(agentID(PortAuthority, addresses(http://193.1.132.98:444/acc)))
BELIEF(fipaMessage(request, PortAuthority, verifyTicket(travellingAgent, 431, ti

-----------COMMITMENTS-----------
COMMIT(Self, Now, BELIEF(true),verifyTicket(PortAuthority, travellingAgent, 43
```

Fig. 4.3. *A TravelOrganiser agent receiving a validation request on a Ticket that was issued by this TravelOrganiser. The agent compares the requested ticket against its records.*

the document creators, Figure 3.1. Each extra security measure that is introduced in this manner, for example a digital certificate, introduces an extra time delay due to the request/present/verify processes.

It is the opinion of the authors that the benefits obtained from imbuing an Agent Factory agent platform with the ATM outweigh the drop in performance and speed. The additional services, such as security, heterogeneity over agent language, behavior, come at a cost. The total time taken for migration to occur and total size of an agent when it is migrated is increased. The modular nature of the ATM however allows for flexibility, for example if speed of migration is a priority, then migration security polices can be set to the lowest levels, increasing performance. If agent platform security is the priority then the resulting increase in the time taken caused by stricter security is an acceptable compromise.

5. Conclusion. This paper has introduced a comprehensive agent migration protocol which it delivers through a set of collaborative intelligent agents. The metaphor has been realized and incorporated within Agent Factory. It represents a consolidation and integration of some previous research that has adopted in part the travel metaphor.

The ATM has been realized in such a way as to support the addition of further modular components and the adoption of configurable local polices, for example baggage allowance, visa issue, security clearance and interoperability between a variety of regimes. This allows agent platform administrators a greater level of management and dynamic control over services provided by the agent platform.

As with the ongoing need for constant vigilance by administrators of computer networks to ever changing threats such as worms, viruses and other malicious attacks, the security threats posed by mobile agents are ever changing. As agents adapt and change so must the agent platforms security countermeasures, the modular service deployment proposed a demonstrated by this paper presents agent platform administrators with the tools necessary to combat and adapt to threats from mobile agents.

Mobile agents also greatly benefit from the use of the ATM. With the introduction of a stable and trusted source from which the agents can learn new behavior, mobile agents can now adapt themselves to perform tasks that would not normally be part of their ordinary operation, as seen when the agent PassportTraveller adapted its behavior in order to fulfil the migration requirements given to it by the PortAuthority agent at its destination, namely presenting a passport, visa and ticket object. Also with the provision of security measures on agent platforms it is possible for mobile agent to access information about agent platforms to verify that a particular platform does not have a history of attacks agents mobile agents, thus allowing agents to protect themselves against malicious hosts.

Acknowledgement. The authors would like to gratefully acknowledge the kind support of Science Foundation Ireland under Grant No. 03/IN.3/I361.

REFERENCES

[1] P. Bellavista, A. Corradi, and C. Stefanelli, *Mobile agent middleware for mobile computing*, j-COMPUTER, 34 (2001), pp. 73–81.

[2] R. H. Bordini and J. A. Campbell, *Anthropologically-based migration of agents: a new approach to interoperability*, 1995.

[3] J. Cao, X. Wang, and S. K. Das, *A framework of using cooperating mobile agents to achieve load sharing in distributed web server groups*, Future Gener. Comput. Syst., 20 (2004), pp. 591–603.

[4] D. Chess, B. Grosof, C. Harrison, D. Levine, C. Parris, and G. Tsudik, *Itinerant agents for mobile computing*, in Readings in Agents, M. N. Huhns and M. P. Singh, eds., Morgan Kaufmann, San Francisco, CA, USA, 1997, pp. 267–282.

[5] D. Chess, C. Harrison, and A. Kershenbaum, *Mobile Agents: Are They a Good Idea?*, Tech. Rep. RC 19887 (December 21, 1994 - Declassified March 16, 1995), IBM, Yorktown Heights, New York, 1994.

[6] R. W. Collier, G. M. P. O'Hare, T. D. Lowen, and C. Rooney, *Beyond prototyping in the factory of agents.*, in Multi-Agent Systems and Applications III, 3rd International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003, Prague, Czech Republic, June 16-18, 2003, Proceedings, vol. 2691 of Lecture Notes in Computer Science, Springer, 2003, pp. 383–393.

[7] K. Decker, K. Sycara, and M. Williamson, *Middle-agents for the internet*, in Proceedings of the 15th International Joint Conference on Artificial Intelligence, Nagoya, Japan, 1997.

[8] F. F. I. P. A. FIPA, *Fipa abstract architecture specification*, June 2002.

[9] S.-U. Guan, T. Wang, and S.-H. Ong, *Migration control for mobile agents based on passport and visa*, Future Gener. Comput. Syst., 19 (2003), pp. 173–186.

[10] W. Jansen, *Countermeasures for mobile agent security*, 2000.

[11] N. R. Jennings, *Agent-oriented software engineering*, in Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World : Multi-Agent System Engineering (MAAMAW-99), F. J. Garijo and M. Boman, eds., vol. 1647, Springer-Verlag: Heidelberg, Germany, 30/6-2/7 1999, pp. 1–7.

[12] N. M. Karnik and A. R. Tripathi, *Design issues in mobile-agent programming systems*, IEEE Concurrency, 6 (1998), pp. 52–61.

[13] S. Poslad and P. Charlton, *Standardizing agent interoperability: the fipa approach*, 2001.

[14] K. Rothermel, F. Hohl, and N. Radouniklis, *Mobile agent systems: What is missing?*, in International Working Conference on Distributed Applications and Interoperable Systems DAISY97, Chapman & Hall, 1997, pp. 111–124.

[15] I. Satoh, *Application-specific routing for mobile agents*, in Proceedings of International Conference on Software Engineering, Artificial Intelligence, Networking & Parallel Distributed Computing, ACIS, August 2001.

[16] J. Tozicka, *Airports for agents: An open mas infrastructure for mobile agents.*, in Multi-Agent Systems and Applications III, 3rd International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003, Prague, Czech Republic, June 16-18, 2003, Proceedings, vol. 2691 of Lecture Notes in Computer Science, Springer, 2003, pp. 373–382.

[17] J. E. White, *Mobile agents.*, in Software Agents, J. Bradshaw, ed., AAAI/MIT Press, Menlo Park, CA, 1997, pp. 437–472.

[18] M. Wooldridge and N. R. Jennings, *Intelligent agents: Theory and practice*, The Knowledge Engineering Review, 10 (1995), pp. 115–152.

Edited by: Shahram Rahimi, Raheel Ahmad
Received: October 28, 2005
Accepted: March 19, 2006

Scalable Computing: Practice and Experience
Volume 7, Number 1, pp. 11–22. http://www.scpe.org
ISSN 1895-1767
© 2006 SWPS

EXPLOITING SHARED ONTOLOGY WITH DEFAULT INFORMATION FOR WEB AGENTS

YINGLONG MA*, BEIHONG JIN†, AND MINGQUAN ZHOU‡

Abstract. When different agents communicate with each other, there needs to be some way to ensure that the meaning of what one agent embodies is accurately conveyed to another agent. It has been argued that ontologies play a key role in communication among different agents. However, in some situations, because there exist terminological heterogeneities and incompleteness of pieces of information among ontologies used by different agents, communication among agents will be very complex and difficult to tackle. In this paper, we proposed a solution to the problem for these situations. We used distributed description logic to model the mappings between ontologies used by different agents and further make a default extension to the DDL for default reasoning. Then, base on the default extension of the DDL model, a complete information query can be reduced to checking default satisfiability of the complex concept corresponding to the query.

Key words. Ontology, Description Logic, Multi-agent System, Satisfiability, Default reasoning.

1. Introduction.

Agents often utilize the services of other agents to perform some given tasks within multi-agent systems [1]. When different agents communicate with each other, there needs to be some ways to ensure that the meaning of what one agent embodies is accurately conveyed to the other agent. Ontologies play a key role in communication among different agents because they provide and define a shared vocabulary about a definition of the world and terms used in agent communication. In real-life scenarios, agents such as Web agents [2] need to interact in a much wider world. The future generation Web, called Semantic Web [3] originates from the form of decentralized vocabularies - ontologies, which are central to the vision of Semantic Web's multi-layer architecture [4]. In the background of the future Semantic Web intelligence, there are terminological knowledge bases (ontologies), reasoning engines, and also standards that make possible reasoning with the marked concepts on the Web. It now seems clear that Semantic Web will not be realized by agreeing on a single global ontology, but rather by weaving together a large collection of partial ontologies that are distributed across the Web [5]. In this situation, the assumption that different agents completely shared a vocabulary is unfeasible and even impossible. In facts, agents will often use private ontologies that define terms in different ways making it impossible for the other agent to understand the contents of a message [6]. Uschold identifies some barriers for agent communication, which can be classified into language heterogeneity and terminological heterogeneity [7]. In this paper, we will focus on terminology heterogeneity and not consider the problem of language heterogeneity.

To overcome these heterogeneity problems, there is a need to align ontologies used by different agents, the most often discussed are merging and mapping of ontologies [8, 9]. However, it seems that the efforts are not enough. For communication among agents with heterogeneous ontologies, there are still some problems that require to be solved. In some situations, only incomplete information can be got. These happen sometime as unavailability of pieces of information, sometime as semantic heterogeneities (here, terminological heterogeneities are focused on) among ontologies from different sources. Another problem is that there always exist some exceptional facts, which conflict with commonsense information. For example, commonly bird can fly, penguin belongs to bird, but penguin couldn't fly. In these situations, communication among agents will be more complex and difficult to tackle. We must consider not only the alignment of ontologies used by different agents, but also the implicit default information hidden among these ontologies. Then, information reasoning for query should be based on both these explicitly represented ontologies and implicit default information. This form of reasoning is called default reasoning, which is non-monotonic. Little attention, however, has been paid to the problem of endowing these logics above with default reasoning capabilities.

For a long time, representation and reasoning in description logic (DL) [10] have been used in a wide range of applications, which are usually given a formal, logic-based semantics. Another distinguished feature is the emphasis on reasoning as a central service. Description logic is very useful for defining, integrating, and

*Computer Sciences and Technology Department, North China Electric Power University, Beijing 102206, P. R. China, m_y_long@otcaix.iscas.ac.cn,
†Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences, P. O. Box 8718, Beijing 100080, P. R. China, jbh@otcaix.iscas.ac.cn,
‡School of Information Sciences, Beijing Normal University, Beijing 100875, P.R.China, mqzhou@bnu.edu.cn

maintaining ontologies, which provide the Semantic Web with a common understanding of the basic semantic concepts used to annotate Web pages. They should be ideal candidates for ontology languages [11]. DAML+OIL [12, 15] and OWL [13] are clear examples of Description Logics. Recently, Borgida and Serafini proposed an extension of the formal framework of description Logic to distributed knowledge models [14], which are called distributed description logic (DDL). A DDL consists of a set of terminological knowledge bases (ontologies) and a set of so-called bridge rules between concept definitions from different ontologies. Two kinds of bridge rules are considered in DDL. Another important feature of DDL is the ability to transform a distributed knowledge base into a global one. In other words, the existing description logic reasoners can be applied for deriving new knowledge.

We adopt the view of [6] that the mappings between ontologies will mostly be established by individual agents that use different available ontologies in order to process a given task. In our opinion, it is a solution to model the mappings between ontologies used by different agents using a DDL and further make a default extension to the DDL for default reasoning. Then, base on the default extension of the DDL model, a complete information query can be reduced to check default satisfiability of the complex concept corresponding to the query.

This paper is organized as follows. Section 2 presents our motivation in making default extension to DDL for communication among multiple agents. Section 3 introduces representation and reasoning related to ontology. Distributed description logic are introduced particularly. In Section 4, we provide a formal framework for default extension to description logic. Default reasoning based on an EDDT is discussed in Section 5. Meanwhile, an algorithm is proposed for checking the default satisfiability of a given concept or a terminological subsumption assertion. Section 6 and Section 7 are related work and conclusion, respectively.

2. Motivation. In order to process a given task in multi-agent systems, it is important and essential to communicate with each other among different agents. However, there often exist some terminological heterogeneities and incompleteness of pieces of information among ontologies used by different agents, which make an agent not completely understand terms used by another agent. In the situations, it is difficult and even impossible to realize the communication among agents. We propose a solution to this problem for the situation. We model the knowledge representation of multi-agents using distributed description logic. The internal mappings between ontologies used by different agents are defined using the so-called bridge rules of distributed description logic. Then, by default extension to the DDL model, we can express explicitly some default information hidden among these ontologies. Based on the extension to DDL model, a query can be reduced to checking the default satisfiability of a concept or an assertion corresponding to the query. More precisely, an adapted algorithm is proposed for checking default satisfiability.

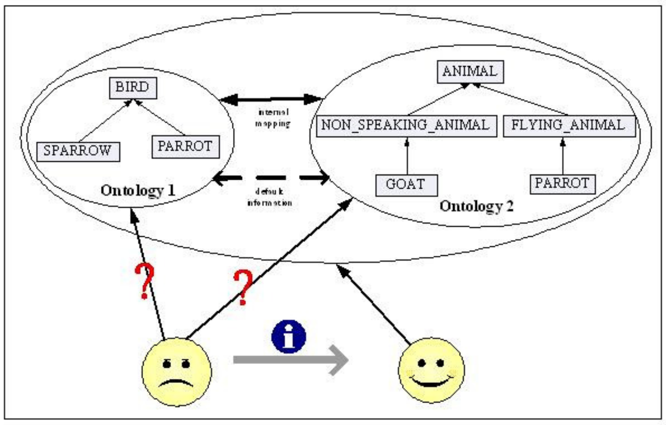


Fig. 2.1. *The situation of communication problem between two agents*

In order to express the problem to be resolved clearer, we make some assumption for simplicity. We only consider communication between two agents, whose ontologies are encoded on the same language. Then, we assume that ontologies used by the two agents have sufficient overlap such that internal mappings between them can be found. The following example shown in Figure 2.1 illustrates the situation described in the paper for our application.

In multi-agent systems, ontologies are used as the explicit representation of domain of interest. To process a given task, an agent perhaps uses multiple ontologies, which usually supplement each other and form a complete model. However, in the model, the default information among these ontologies is not considered. For example, we perhaps establish the internal mapping specifying that BIRD is a subclass of NON_SPEAKING_ANIMAL. Through the agent using ontology 1, we take the following query BIRD$\wedge$NON_SPEAKING_ANIMAL. The found question is that the agent using ontology 1 doesn't know the meaning of the term

NON_SPEAKING_ANIMAL

which only can be understood by the agent using ontology 2. To get complete and correct results of query, the two agents must coordinate each other. Another question is that the query results of the agent will include SPARROW and PARROT. We will find the results are partially correct because the class of PARROT can speak like man. The reason getting partially correct results is that we have not considered the default fact: in most cases, birds cannot speak; parrots belong to the class of birds but they can speak. In our opinion, default information should be considered and added into the model with multiple ontologies, which will form a sufficiently completely model. Then, available reasoning support for ontology languages is based on the model with default information.

3. Representation and Reasoning Related to Ontologies. A formal and well-founded ontology language is the basis for knowledge representation and reasoning about the ontologies involved. Description Logic is a formalism for knowledge representation and reasoning. Description logic is very useful for defining, integrating, and maintaining ontologies, which provide the Semantic Web with a common understanding of the basic semantic concepts used to annotate Web pages. It should be ideal candidates for ontology languages. One of the important proposals that have been made for well-founded ontology languages for the Web is DAML+OIL. Recently, description logic has heavily influenced the development of the semantic Web language. For example, DAML+OIL ontology language is just an alternative syntax for very expressive description logic [12]. So in the following sections, we use syntax and semantic representations of description logic involved instead of DAML+OIL. Description Logics is equipped with a formal, logic-based semantics. Its another distinguished feature is the emphasis on reasoning as a central service.

3.1. Description Logic. The basic notations in DL are the notation of concepts embracing some individuals on a domain of individuals, and roles representing binary relations on the domain of individuals. A specific DL provides a specific set of constructors for building more complex concepts and roles. For examples:

— the symbol $\top$ is a concept description which denotes the top concept, while the symbol $\bot$ stands for the inconsistent concept which is called bottom concept.

— the symbol $\sqcap$ denotes concept conjunction, e. g., the description Person$\sqcap$Male denotes the class of man.

— the symbol $\forall$R.C denotes the universal roles quantification (also called value restriction), e. g., the description $\forall$hasChild.Male denotes the set of individual whose children are all male.

— the number restriction constructor ($\geq$nR.C) and ($\leq$nR.C), e. g., the description ($\geq$1 hasChild.Doctor) denotes the class of parents who have at least one children and all the children are doctors.

The various description logics differ from one to another based on the set of constructors they allow. Here, we show the syntax and semantics of $\mathcal{ALCN}$ [16], which are listed as Figure 3.1.

Then we can make several kinds of assertions using these descriptions. There exist two kinds of assertions: subsumption assertions of the form $C \sqsubseteq D$ and assertions about individuals of the form $C(a)$ or $p(a, b)$, where C and D denote Concepts, p denotes role, and a and b are individual, respectively. For examples, the assertion $Parent \sqsubseteq Person$ denotes the fact the class of parents is subsumed by the class of person. The description $Person(a)$ denotes that the individual a is a person while the description $hasChild(a, b)$ denotes a has a child who is b. The collection of subsumption assertions is called Tbox, which specifies the terminology used to describe some application domains. A Tbox can be regarded as a terminological knowledge base of the description logic.

An interpretation for DL $\mathcal{I} = (\Delta^{\mathcal{I}}, \bullet^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a domain of objects and $\bullet^{\mathcal{I}}$ the interpretation function. The interpretation function maps roles into subsets of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, concepts into subsets of $\Delta^{\mathcal{I}}$ and individuals into elements of $\Delta^{\mathcal{I}}$. Satisfactions and entailments in DL Tbox will be described using following notations:

— $\mathcal{I} \models C \sqsubseteq D$ iff $C^{\mathcal{I}} \sqsubseteq D^{\mathcal{I}}$

— $\mathcal{I} \models T$, iff for all $C \sqsubseteq D$ in T, $\mathcal{I} \models C \sqsubseteq D$

— $C \sqsubseteq D$, iff for all possible interpretations $\mathcal{I}$, $\mathcal{I} \models C \sqsubseteq D$

DL Syntax	**DL Semantic**
$\neg C$	$\mathbf{L} \backslash C^{\mathcal{I}}$
$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
$\exists P.C$	$\{x \vert \exists y.(x,y) \in P^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
$\forall R.C$	$\{x \vert \forall y.(x,y) \in P^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
$P \sqsubseteq R$	$P^{\mathcal{I}} \subseteq R^{\mathcal{I}}$
$C \sqsubseteq \neg D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$
$\geq nP.C$	$\{x \in \mathbf{L} \mid \Vert \{y \in \mathbf{L} \vert (x,y) \in P^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \Vert \geq n\}$
$\leq nP.C$	$\{x \in \mathbf{L} \mid \Vert \{y \in \mathbf{L} \vert (x,y) \in P^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \Vert \leq n\}$
$C(a)$	$a \in C^{\mathcal{I}}$
$P(a,b)$	$(a,b) \in P^{\mathcal{I}}$

FIG. 3.1. *Syntax and semantics of ontology representation*

— $T \models C \sqsubseteq D$, iff for all interpretations $\mathcal{I}$, $\mathcal{I} \models C \sqsubseteq D$ such that $\mathcal{I} \models T$

3.2. Distributed Description Logic. A DDL is composed of a collection of “distributed" DLs, each of which represents a subsystem of the whole system. All of DLs in DDL are not completely independent from one another as the same piece of knowledge might be presented from different points of view in different DLs. Each DL autonomously represents and reasons about a certain subset of the whole knowledge. Distributed description logic (DDL) can better present heterogeneous distributed systems by modeling relations between objects and relations between concepts contained in different heterogeneous ontologies.

A DDL consists of a collection of DLs, which is written $\{DL_i\}_{i \in \mathrm{I}}$, every local DL in DDL is distinguished by different subscripts. The constraint relations between different DLs are described by using so-called “bridge rules" in an implicit manner, while the constraints between the corresponding domains of different DLs are described by introducing the so-called “semantics binary relations". In order to support directionality, the bridge rules from DL_i to DL_j will be viewed as describing “flow of information" from DL_i to DL_j from the point of view of DL_j. In DDL, $i : C$ denotes the concept C in DL_i, $i : C \sqsubseteq D$ denotes subsumption assertion $C \sqsubseteq D$ in DL_i. A bridge rule from i to j is described according to following two forms: $i : C \xrightarrow{\sqsubseteq} j : D$ and $i : C \xrightarrow{\sqsupseteq} j : D$. The former is called into-bridge rule, and the latter called onto-bridge rule. A DDL embraces a set of subsumption assertions, which are called DTB. A distributed Tbox (DTB) is defined based on Tboxes in all of local DLs and bridge rules between these Tboxes. A DTB $DT = (\{T_i\}_{i \in \mathrm{I}}, B)$, where T_i is Tbox in DL_i, and for every $i \neq j \in \mathrm{I}$, $B = \{B_{ij}\}$, where B_{ij} is a set of bridge rules from DL_i to DL_j. A DTB can be regarded as a distributed terminological knowledge base for the distributed description logics.

The semantics for distributed description logics are provided by using local interpretation for individual DL and connecting their domains using semantics binary relations r_{ij}. A distributed interpretation $\mathcal{J}=(\{\mathcal{I}_i\}_{i \in \mathrm{I}}, r)$ of DT consists of interpretations $\mathcal{I}_i$ for DL_i over domain $\Delta^{\mathcal{I}_i}$, and a function r associating to each $i, j \in \mathrm{I}$ a binary relation $r_{ij} \subseteq \Delta^{\mathcal{I}_i} \times \Delta^{\mathcal{I}_j}$. $r_{ij}(d) = \{d' \in \Delta^{\mathcal{I}_j} \vert (d, d') \in r_{ij}\}$, and for any $D \in \Delta^{\mathcal{I}_j}$, $r_{ij}(D) = \cup_{d \in D} r_{ij}(d)$. Note that semantic relation r must be bold everywhere.

A distributed interpretation $\mathcal{J}$ d-satisfies (written $\models_d$) the elements of DTB $DT = (\{T_i\}_{i \in \mathrm{I}}, B)$ according to following clauses: For every $i, j \in \mathrm{I}$

— $\mathcal{J} \models_d i : C \xrightarrow{\sqsubseteq} j : D$ if $r_{ij}(C^{\mathcal{I}_i}) \subseteq D^{\mathcal{I}_j}$

— $\mathcal{J} \models_d i : C \xrightarrow{\sqsupseteq} j : D$ if $r_{ij}(C^{\mathcal{I}_i}) \supseteq D^{\mathcal{I}_j}$

— $\mathcal{J} \models_d i : C \sqsubseteq D$ if $\mathcal{I}_i \models C \sqsubseteq D$

— $\mathcal{J} \models_d T_i$, if for all $C \sqsubseteq D$ in T_i such that $\mathcal{I}_i \models C \sqsubseteq D$

— $\mathcal{J} \models_d DT$, if for every $i, j \in \mathrm{I}$, $\mathcal{I}_i \models_d T_i$ and $\mathcal{I}_i \models_d b$, for every $b \in \cup B_{ij}$

— $DT \models_d i : C \sqsubseteq D$, if for every distributed interpretation $\mathcal{J}$, $\mathcal{J} \models_d DT$ implies $\mathcal{J} \models_d i : C \sqsubseteq D$

4. Default Extension to DDL. DDL is used to better model knowledge representation in a multi-agent systems, where ontologies are used as the explicit representation of domain of interest. The internal mappings

DT={{T_1={PARROT⊑BIRD,SPARROW⊑BIRD},
T_2={PARROT⊑FLYING_ANIMAL,
GOAT⊑ ¬SPEAKING_ANIMAL}
B={1:PARROT⊑2:PARROT}}

DF={BIRD(x):PARROT(x)/¬SPEAKING_ANIMAL(x)}

Fig. 4.1. *DT and D of the DDT*

$\Delta^{\mathcal{I}_1}$={parrot1,parrot2,sparrow,swan}, PARROT$^{\mathcal{I}_1}$={parrot1,parrot2}
SWAN$^{\mathcal{I}_1}$={swan}, SPARROW$^{\mathcal{I}_1}$={sparrow}
BIRD$^{\mathcal{I}_1}$={parrot1,parrot2,sparrow,swan}

$\Delta^{\mathcal{I}_2}$={parrot,goat,butterfly}, PARROT$^{\mathcal{I}_2}$={parrot}
GOAT$^{\mathcal{I}_2}$={goat}, FLYING_ANIMAL$^{\mathcal{I}_2}$={parrot,butterfly}
¬SPEAKING_ANIMAL$^{\mathcal{I}_2}$={goat},

r_{12}={(parrot1, parrot), (parrot2, parrot)}

Fig. 4.2. *The distributed interpretation of the DDT*

between ontologies used by different agents are defined using the so-called bridge rules of distributed description logic. As mentioned in Section 2, however, DDL model is not sufficient for modeling communication among multiple agents with heterogeneous ontologies because the default information among these ontologies is not considered. In this situation, query based on multi-agent systems will be possible to get partially correct results. To construct a sufficiently completely model, default information should be considered and added into the DDL model with multiple ontologies. In the following, we discuss the problem of default extension to DDL.

Our default extension approach is operated on a distributed terminological knowledge base. A distributed terminological knowledge base originally embraces only some strict information (i. e., the information having been expressed explicitly in distributed terminological knowledge base). Default information is used for getting complete and correct information from multiple distributed ontologies. We should consider a way to explicitly include and express the default information in a distributed terminological knowledge base for reasoning based on these distributed ontologies. To be able to include default information in distributed knowledge base, we firstly introduce the notation description of a default rule.

Definition 4.1. *A default rule is of the form $P(x) : J_1(x), J_2(x), \cdots, J_n(x)/C(x)$, where P, C and J_i are concept names ($1 \leq i \leq n$), and x is a variable. $P(x)$ is called the prerequisite of the default, all of $J_i(x)$ are called the justifications of the default, and $C(x)$ is called the consequent of the default. The meaning of default rule $P(x) : J_1(x), J_2(x), \cdots, J_n(x)/C(x)$ can be expressed as follows:*

If there exists an interpretation $\mathcal{I}$ such that $\mathcal{I}$ satisfies $P(x)$ and doesn't satisfy every $J_i(x)$ ($1 \leq i \leq n$), then $\mathcal{I}$ satisfies $C(x)$. Otherwise, if $\mathcal{I}$ satisfies every $J_i(x)$ ($1 \leq i \leq n$), then $\mathcal{I}$ satisfies $C(x)$.

For example, to state that a person can speak except if s/he is a dummy, we can use the default rule

Person(x):Dummy(x)/CanSpeak(x).

If there is an individual named John in a domain of individuals, then the closed default rule is

Person(John):Dummy(John)/CanSpeak(John).

To deal with strict taxonomies information as well as default information in distributed knowledge base, the definition of distributed knowledge base should be extended for including a set of default rules. We call the distributed terminological knowledge base with explicit default information default distributed terminological knowledge base, which is denoted as DDT.

Definition 4.2. *A default distributed terminological knowledge base DDT=(DT,D), where DT is the DTB of distributed description logic, and D is a set of default rules.*

An example of a DDT is shown in figure 4.1. The DT of the DDT is based on two local terminological knowledge bases, named T_1 and T_2 respectively. The DT and D of the DDT are shown in Figure 4.1. Figure 4.2 provides a distributed interpretation of the DDK.

The satisfaction problem of DDT should be discussed for queries based on it. The satisfaction symbol is denoted as $\models_{dd}$. The kind of satisfiability of these elements in DDT means that they should satisfy not only DT, but also the set D of default rules. So we call satisfiability of elements in DDT default satisfiability. Default satisfiability serves as a complement of satisfiability definition in a distributed terminological knowledge base with default rules. In queries based on DDT, the definition will be used to detect satisfiability of a concept or assertion.

DEFINITION 4.3. *A distributed interpretation $\mathcal{J}$ dd-satisfies (written $\models_{dd}$) the elements of $DDT = (DT, D)$, according to following clauses: For every default rule δ in D, δ=$P(x) : J_1(x), J_2(x), \ldots, J_n(x)/C(x)$, for every $i, j \in$ I*

— $\mathcal{J} \models_{dd} DDT$, if $\mathcal{J} \models_d DT$ and $\mathcal{J} \models_d \delta$

— $\mathcal{J} \models_{dd} DT$, if $\mathcal{J} \models_d DT$ and $\mathcal{J} \models_d \delta$

— $\mathcal{J} \models_d \delta$, if $\mathcal{J} \models_d P \sqsubseteq C$ implies $\mathcal{J} \not\models_d J_k \sqsubseteq \neg C$ for all k $(1 \leq k \leq n)$

— $\mathcal{J} \models_d P \sqsubseteq C$, if $i \neq j$, such that $\mathcal{J} \models_d i : P \sqsubseteq C$ or $\mathcal{J} \models_d i : P \xrightarrow{\sqsubseteq} j : C$ or $\mathcal{J} \models_d j : C \xrightarrow{\sqsupseteq} i : P$

— $DDT \models_{dd} DT$, if for all distributed interpretation $\mathcal{J}$, $\mathcal{J} \models_{dd} DDT$ implies $\mathcal{J} \models_{dd} DT$

In a distributed knowledge base, default information may have been used during reasoning, but a DDT is not really helpful for reasoning with default information in distributed knowledge. Some additional information with respect to default rules should be included explicitly into DT. A closed default rule of the form $P(x)$: $J_1(x), J_2(x), \cdots, J_n(x)/C(x)$ can be divided into two parts: $P(x) \rightarrow C(x)$ and $J_i(x) \rightarrow C(x)$, $(1 \leq i \leq n)$. We call the first part fulfilled rule, and the second exceptional rules. A rule of the form $A(x) \rightarrow B(x)$ means for every (distributed) interpretation $\mathcal{I}$, $x \in A^{\mathcal{I}}$, then $x \in B^{\mathcal{I}}$, i. e. $A \sqsubseteq B$, where A and B are concept names, and x denotes an individual.

DEFINITION 4.4. *An extended distributed knowledge base EDDT is constructed based on a DDT=(DT,D), according to the following clauses: For every default rule δ in D,δ=$P(x) : J_1(x), J_2(x), \ldots, J_n(x)/C(x)$,*

1) Dividing into two parts which embrace fulfilled rules and exceptional rules, respectively. The fulfilled rule denotes that it holds in most cases until the exception facts appear, while the exceptional rules denote some exceptional facts.

2) Adding $P \sqsubseteq C$ and $J_i \sqsubseteq C$ into DT $(1 \leq i \leq n)$, which are the assertions corresponding to fulfilled rule and exceptional rules, respectively

3) Setting the priorities of different rules for selecting appropriate rules during reasoning. The assertions corresponding to exceptional rules have the highest priority, while original strict information has normal priority. The assertions corresponding to fulfilled rules are given the lowest priority.

In the course of constructing an EDDT, default information has been added into distributed knowledge base for default reasoning, because these default information may have been used during reasoning. Exceptional information has been assigned the highest priority to avoid conflicting with some strict information, while fulfilled rules would be used only in the situation that no other strict information can be used, its priority is least. A simplified view of the EDDT based on the DDT and its interpretation (shown in figure 4.1 and 4.2) can be found in figure 4.3. The default rule $BIRD(x) : PARROT(x)/SPEAKING\_ANIMAL(x)$ is divided into one fulfilled rule and one exceptional rule, the fulfilled rule $BIRD \sqsubseteq \neg SPEAKING\_ANIMAL$ and the exceptional rule $PARROT \sqsubseteq SPEAKING\_ANIMAL$ has been added into EDDT. In fact, an EDDT can be recognized as a collection of integrated ontologies with default information expressed explicitly. Default reasoning can be performed based on an EDDT. In the following section, we will focus on how the default reasoning based on EDDT will be realized. Meanwhile, an adapted algorithm will be discussed for checking default satisfiability of complex concepts and subsumption assertions.

5. Reasoning with Default Information.

Reasoning with default information provides agents using different ontologies with stronger query capability. In our opinion, a query based on DDT can boil down to checking default satisfiability of complex concept in accord with the query. Based on description logics, satisfiability of a complex concept is decided in polynomial time according to Tableau algorithm for $\mathcal{ALCN}$ [10, 16]. An important result of DDL is the ability to transform a distributed knowledge base into a global one. So the existing description logic reasoners can be applied for deriving new knowledge. This would allow us to transfer theoretical results and reasoning techniques from the extensive current DL literatures. In our reasoning approach with default information, the result will be used. The reasoning problem of distributed terminological

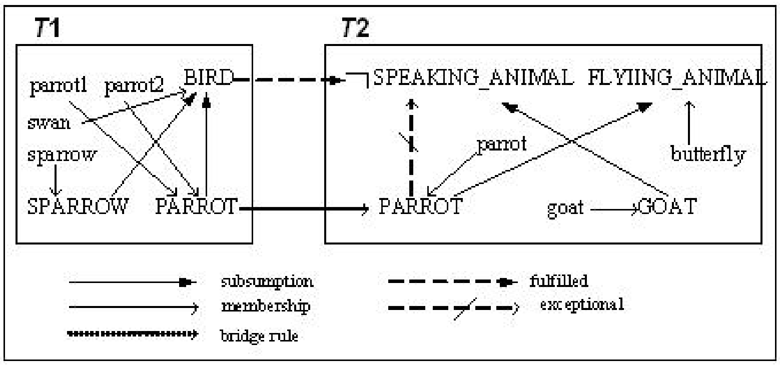


Fig. 4.3. *An example of EDDT*

knowledge base of a DDL will be transformed to the reasoning problem of terminological knowledge base of a global DL corresponding to the DDL. So in our opinion, detecting default satisfiability of a DDL is just detecting the default satisfiability of the global DL in accord with the DDL. A default extension to Tableau algorithm for $\mathcal{ALCN}$ DL can be used for detecting default satisfiability of $\mathcal{ALCN}$ concepts based on an EDDT.

Definition 5.1. *A constraint set S consists of constraints of the form C(x), p(x,y), where C and p are concept name and role name, respectively. Both x and y are variables.*

An $\mathcal{I}$-assignment maps a variable x into a element of $\Delta^{\mathcal{I}}$. If $x^{\mathcal{I}} \in C^{\mathcal{I}}$, the $\mathcal{I}$-assignment satisfies $C(x)$. If $(x^{\mathcal{I}}, y^{\mathcal{I}}) \in p^{\mathcal{I}}$, the $\mathcal{I}$-assignment satisfies $p(x, y)$. If the $\mathcal{I}$-assignment satisfies every element in constraint set S, it sati sfies S. If there exist an interpretation $\mathcal{I}$ and an $\mathcal{I}$-assignment such that the $\mathcal{I}$-assignment satisfies the constraint set S, S is satisfiable. S is satisfiable iff all the constraints in S are satisfiable.

It will be convenient to assume that all concept descriptions in EDDT are in *negation normal form* (NNF). Using de-Morgan's rules and the usual rules for quantifiers, any $\mathcal{ALCN}$ concept description can be transformed into an equivalent description in NNF in linear time. For example, the assertion description $SPARROW \sqsubseteq BIRD$ can be transformed the form $\neg SPARROW \sqcup BIRD$. To check satisfiability of concept C, our extended algorithm starts with constraint set $S = \{C(x)\}$, and applies transformation rules in an extended distributed knowledge base. The concept C is satisfiable iff the constraint set S is unsatisfiable. In applying transformation rules, if there exist all obvious conflicts (clashes) in S, S is unsatisfiable, which means the concept C is satisfiable. Otherwise, S is unsatisfiable. The transformation rules are derived from concepts and assertions in EDDT. If the constraint set S before the action is satisfiable, S after the action is also satisfiable. The transformation rules of default extension to satisfiability algorithm are shown as Figure 5.1.

When the adapted algorithm is used for detecting default satisfiability of $\mathcal{ALCN}$ concepts, every action must preserve satisfiability. Because if an action don't preserve satisfiability, we cannot ensure the condition that if the constraint set before the action is satisfiable then the set after the action is satisfiable. In the extension algorithm, we must prove the actions preserve satisfiability.

Theorem 5.2. *The action of the applied transformation rules preserves satisfiability.*

Proof. Because a DDL can be regarded as a global DL, for simplification, we use interpretation $\mathcal{I}$ of the global DL for distributed interpretation $\mathcal{J}$ of the DDL.

In the extension algorithm, every step may involve the actions of some transformation rules that are applied. so we must prove all of these actions in these steps preserve satisfiability. Because the actions in the second step are originally derived from the classical Tableau algorithm, we have known they preserve satisfiability [10]. The remainder of the proof will only consider the actions in the first step and the third step.

1) In the first step, the action condition is that for any default rule of the form

$$\mathrm{P}(x)\colon \mathrm{J}_1(x), \mathrm{J}_2(x), \ldots, \mathrm{J}_n(x)/\mathrm{C}(x)$$

in set of default rules, there exists $\mathrm{J}_i(x)$ is contained in S. If the constraint set S before the action is satisfiable, then there exists an interpretation $\mathcal{I}$ such that $\mathcal{I}$ satisfies all of elements of S. Because $\{\mathrm{J}_i(x)\}\subseteq$S, then $\mathcal{I}$ satisfies $\mathrm{J}_i(x)$ $(1\leqslant i \leqslant$n). Furthermore, according to the Definition 4.1, we know $\mathcal{I}$ satisfies $\neg\mathrm{C}(x)$ after the action. From the above, we know that $\mathcal{I}$ satisfies both $\neg\mathrm{C}(x)$ and S, i. e., $\mathcal{I}$ satisfies S$\cup\{\neg\mathrm{C}(x)\}$ after the action.

Exceptional rules:(Used for Step 1)
Condition:
For any default rule of the form P(x):J$_1$(x), J$_2$(x),…,J$_n$(x)/C(x), there exists J$_i$(x) ($1\leqslant i \leqslant$n) is contained in S,but S doesn't contain $\neg$C(x).
Action:
S=S$\cup\{\neg$C(x)$\}$
Strict rules:(Used for Step 2)
$\sqcap-$rule:
Condition:
$\{$(C$\sqcap$D)(x)$\}\subseteq$S, but S doesn't contain both C(x) and D(x).
Action:
S=S$\cup\{$C(x), D(x)$\}$
$\sqcup-$rule:
Condition:
$\{$(C$\sqcup$D)(x)$\}\subseteq$S but $\{$C(x),D(x)$\}\cap$ S=$\emptyset$.
Action:
S=S$\cup\{$C(x)$\}$ or S=S$\cup\{$D(x)$\}$
$\exists-$rule:
Condition:
$\{(\exists$R.C)(x)$\}\subseteq$S, but there is no individual name y such that S contains C(x) and R(x, y).
Action:
S=S$\cup\{$C(y), R(x, y)$\}$
$\forall-$rule:
Condition:
$\{(\forall$R.C)(x), R(x, y)$\}\subseteq$S, but S doesn't contain C(y).
Action:
S=S$\cup\{$C(y)$\}$
$\geq$n-rule:
Condition:
($\geq$nR)(x) S, there doesn't exist individual names y$_1$, y$_2$,$\cdots$,y$_n$ such that R(x,y$_i$) and y$_i\neq$y$_j$ are in S, ($1\leq$i$\leq$j$\leq$n).
Action:
S=S$\cup$R(x,y$_i$)$\cup$y$_i\neq$y$_j$, ($1\leq$i$\leq$j$\leq$n), where y$_1$,$\cdots$, y$_n$ are distinct individual names not occurring in S.
$\leq$n-rule:
Condition:
distinct individual names y$_1$,$\cdots$,y$_{n+1}$ are contained in S such that ($\leq$nR)(x) and R(x,y$_1$),$\cdots$, R(x,y$_{n+1}$) are in S, and y$_i\neq$y$_j$ is not in S for some i,j,
$1\leq$i$\leq$j$\leq$n+1.
Action:
for each pair y$_i$ and y$_j$, such that $1\leq$i$\leq$j$\leq$n+1 and yi$\neq$yj is not in S, the S$_{i,j}$:=[y$_i$/y$_j$]S is obtained from S by replacing each occurrence of y$_i$ by y$_j$.
Fulfilled rule: (Used for Step 3)
Condition:
no other transformation rules is applicable, and for any default rule of the form P(x):J$_1$(x), J$_2$(x), …, J$_n$(x)/C(x), $\{$P(x)$\}\subseteq$S, but all of the J$_i$(x) ($1\leqslant i \leqslant$n) and C(x) are not contained in S.
Action:
S=S$\cup\{$C(x)$\}$

FIG. 5.1. *The adapted Tableau rules used for detecting default satisfiability of $\mathcal{ALCN}$ concepts*

2) In the third step, the action condition is that $\{$P(x)$\}\subseteq$S, S doesn't contain all of the J$_i$(x) ($1\leqslant i \leqslant$n) and no other transformation rules can be applied. If the constraint set S before the action is satisfiable, then there exists an interpretation $\mathcal{I}$ such that $\mathcal{I}$ satisfies all of elements of S. Because $\{$P(x)$\}\subseteq$S, then $\mathcal{I}$ satisfies P(x). Furthermore, we know that $\mathcal{I}$ doesn't satisfy any J$_i$(x) ($1\leqslant i \leqslant$n), otherwise, there would exist other exceptional rules which can be applied. Because $\mathcal{I}$ satisfies P(x) before the action. So from Definition 4.1, we get $\mathcal{I}$ satisfies C(x). Because $\mathcal{I}$ satisfies both S and C(x), we get $\mathcal{I}$ satisfies S$\cup\{$C(x)$\}$.

From above proofs, we can conclude that every action in the applied transform rules, in the extension algorithm, preserves satisfiability. □

As mentioned in Definition 4.4, an EDDT embraces three types of transformation rules: strict information, fulfilled information and exceptional information. These different types of information are given different levels of priority. Here, we use the symbol SR to denote the set of strict facts in an EDDT, FR to denote the set of fulfilled information and ER to denote the set of exceptional information. Then, based on the EDDT shown in Figure 4.3, we will get the descriptions of its sets of different types of information in NNF, where

Algorithm: checking default satisfiability of C based on the EDDT

```
Require: An EDDT which embraces SR, FR and ER.
Ensure: the descriptions of SR, FR and ER in NNF.
1. $S_0=C(x)$, i=1;
2. apply strict rules and transform $S_0$ into $S_i$;
3. for each r∈ER do                 //Step 1
4.    if $S_i$ meets the condition of r
5.       apply r to $S_i$ and result of action: $S_{i+1}\leftarrow S_i$;
6.       i=i+1;
7.       if there exist clashes in $S_i$
8.          return "C is satisfiable";
9.       end if
10.   end if
11. end for
12. for each r∈SR do                // Step 2
13.   if $S_i$ meets the condition of r and $S_i$ isn't labeled "Clash"
14.      apply r to $S_i$ and result of action: $S_{i+1}\leftarrow S_i$;
15.      i=i+1;
16.      if there exist clashes in $S_i$
17.         $S_i$ is labeled "Clash";
18.      end if
19.   end if
20. end for
21. for each r∈FR do                // Step 3
22.   if $S_i$ meets the condition of r and $S_i$ isn't labeled "Clash"
23.      apply r to $S_i$ and result of action: $S_{i+1}\leftarrow S_i$;
24.      i=i+1;
25.      if there exist clashes in $S_i$
26.         $S_i$ is labeled "Clash";
27.      end if
28.   end if
29. end for
30. if the leaf nodes of all possible branches in the constructed tree-like model are labeled "Clash"
31.   return "C is satisfiable";
32. else return "C is unsatisfiable";
33. end if
```

$$\begin{aligned}SR = \{&\neg PARROT \sqcup BIRD, \neg SPARROW \sqcup BIRD,\\ &\neg PARROT \sqcup FLYING\_ANIMAL, \neg GOAT \sqcup \neg SPEAKING\_ANIMAL\}\\ FR = \{&\neg BIRD \sqcup \neg SPEAKING\_ANIMAL\}\\ ER = \{&\neg PARROT \sqcup SPEAKING\_ANIMAL\}.\end{aligned}$$

The subsumption assertions to be checked should be transformed into their negation description in NNF according to the theorem [10]: $A \sqsubseteq B$ is satisfiable iff $A \sqcap \neg B$ is unsatisfiable, where A and B are concept descriptions, respectively. For example, the subsumption assertion $SPARROW \sqsubseteq \neg SPEAKING\_ANIMAL$ will be transformed into the concept description with negation $SPARROW \sqcap SEAKING\_ANIMAL$. In the following, we will describe particularly the extension algorithm for checking default satisfiability of a given concept. The default extension algorithm can be divided into three steps. In the first step, we apply exceptional rules to constraint set because they have the highest priority. If exceptional rules can be used for the detected concept, strict rules will not be used. Otherwise, if no exceptional rules can be used, the strict rules can be applied to constraint set (step 2). The reason why we do like this is to avoid conflicting with some strict information. Another reason is to save reasoning time. In step three, only in the situation that no other strict information can be used, could fulfilled rules be used. The default extension algorithm either stops because all actions fail with obvious conflicts, or it stops without further used rules.

The following example shown in Figure 5.2 demonstrates the algorithm with a tree-like diagram. We want to know whether the subsumption assertion $SPARROW \sqsubseteq \neg SPEAKING\_ANIMAL$ is satisfiable in the EDDT shown in Figure 4.3. That is to say, we should detect that the concept $SPARROW \sqcap SPEAKING\_ANIMAL$ is unsatisfiable. The concept is firstly transformed into constrain set S_0. Considering the default rule $BIRD(x)$:

$PARROT(x)/SPEAKING\_ANIMAL(x)$, we know that $PARROT(x)$ isn't contained in S_0, Then, in the first step, the exceptional rule $\neg PARROT(x) \sqcup SPEAKING\_ANIMAL(x)$ can not be applied to S_0. In the following steps, we apply strict rules, the reasoning continues until it stops with obvious conflicts. Finally, the leaf node of every branch in this tree-like diagram is notated using "Clash" tag. So we know the constraint $SPARROW \sqcap SPEAKING\_ANIMAL$ are not satisfiable. That is to say, the subsumption assertion $SPARROW \sqsubseteq \neg SPEAKING\_ANIMAL$ is satisfiable.

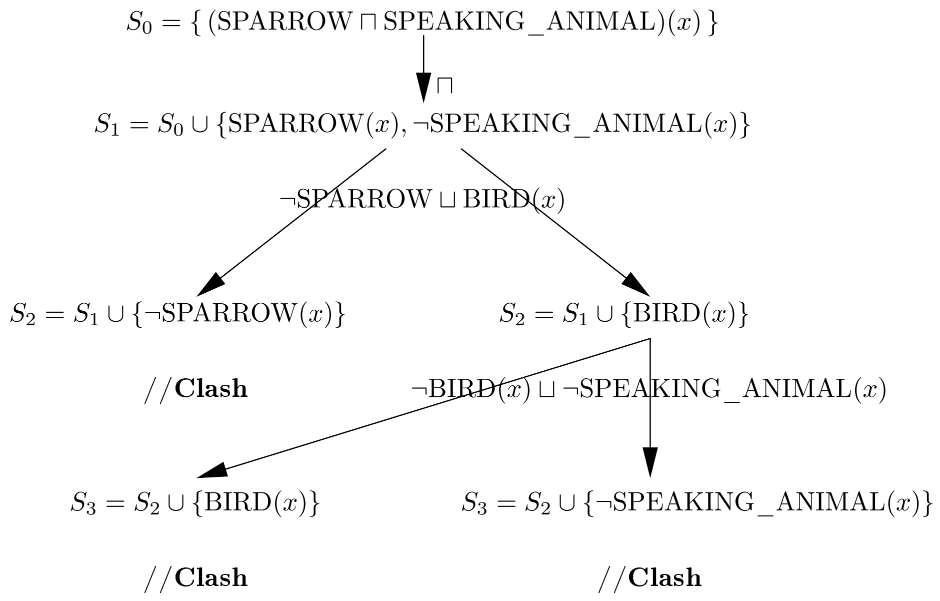

$S_0 = \{\,(\mathrm{SPARROW} \sqcap \mathrm{SPEAKING\_ANIMAL})(x)\,\}$
$\sqcap$
$S_1 = S_0 \cup \{\mathrm{SPARROW}(x), \neg\mathrm{SPEAKING\_ANIMAL}(x)\}$
$\neg\mathrm{SPARROW} \sqcup \mathrm{BIRD}(x)$
$S_2 = S_1 \cup \{\neg\mathrm{SPARROW}(x)\}$
$S_2 = S_1 \cup \{\mathrm{BIRD}(x)\}$
//**Clash**
$\neg\mathrm{BIRD}(x) \sqcup \neg\mathrm{SPEAKING\_ANIMAL}(x)$
$S_3 = S_2 \cup \{\mathrm{BIRD}(x)\}$
$S_3 = S_2 \cup \{\neg\mathrm{SPEAKING\_ANIMAL}(x)\}$
//**Clash**
//**Clash**

FIG. 5.2. *Detecting default satisfiability of complex concept*

Please note that the extension algorithm can tackle both general subsumption assertions and assertions about exceptional facts. In another example shown in Figure 5.3, we want to check whether the subsumption assertion $PARROT \sqsubseteq SPEAKING\_ANIMAL$ is satisfiable, that is to say, we check the default satisfiability of the concept $PARROT(x) \sqcap \neg SPEAKING\_ANIMAL(x)$, which transformed into a constrain set. In the first step, when the exceptional rule $\neg PARROT(x) \sqcup SPEAKING\_ANIMAL(x)$ is applied to constraint set, the complete conflicts occur. So we know the concept $PARROT(x) \sqcap \neg SPEAKING\_ANIMAL(x)$ is not satisfiable, which means that the subsumption assertion $PARROT \sqsubseteq SPEAKING\_ANIMAL$ is satisfiable. Then reasoning process stops without applying other transformation rules. This can be served as an example of reasoning for an exceptional fact.

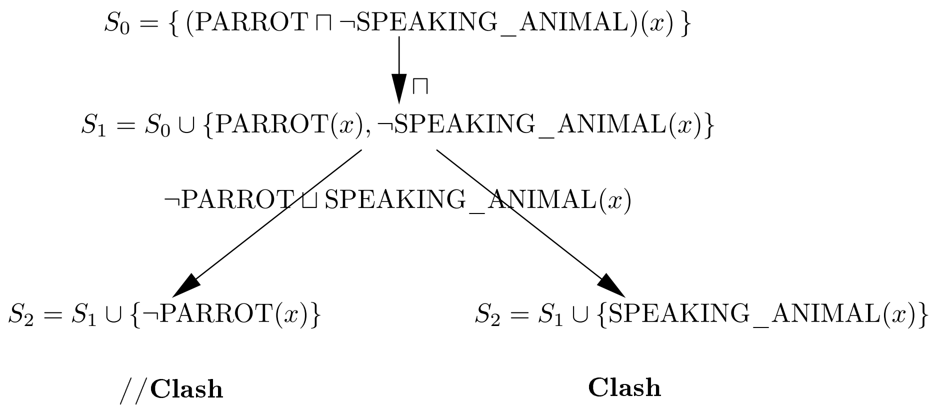

$S_0 = \{\,(\mathrm{PARROT} \sqcap \neg\mathrm{SPEAKING\_ANIMAL})(x)\,\}$
$\sqcap$
$S_1 = S_0 \cup \{\mathrm{PARROT}(x), \neg\mathrm{SPEAKING\_ANIMAL}(x)\}$
$\neg\mathrm{PARROT} \sqcup \mathrm{SPEAKING\_ANIMAL}(x)$
$S_2 = S_1 \cup \{\neg\mathrm{PARROT}(x)\}$
$S_2 = S_1 \cup \{\mathrm{SPEAKING\_ANIMAL}(x)\}$
//**Clash**
Clash

FIG. 5.3. *An example of detecting exceptional fact*

In the following, we give a brief of discussion of complexity issues about the default satisfiability algorithm.

THEOREM 5.3. *Default satisfiability of $\mathcal{ALCN}$-concept descriptions is PSPACE-complete.*

Proof. From [16], we know that satisfiability of $\mathcal{ALCN}$-concept descriptions is PSPACE-complete. As mentioned above, our default satisfiability algorithm for $\mathcal{ALCN}$-concept descriptions can be divided into three steps. In fact, every step is just the satisfiability algorithm for $\mathcal{ALCN}$. Then the sequence of the three steps is also essentially the satisfiability algorithm for $\mathcal{ALCN}$. So we get the conclusion that default satisfiability of $\mathcal{ALCN}$-concept descriptions is PSPACE-complete. □

6. Related work and Discussions. In the description logics community, a number of approaches to extend description logics with default reasoning have been proposed. Baader and Hollunder [17] investigated the problems about open default in detail and defined a preference relation. The approach is not restricted to simple normal default. Two kinds of default rules were introduced by Straccia [18]. The first kind is similar to the fufilled rules in our approach. The second kind of rules allows for expressing default information of fillers of roles. Lambrix [19] presented a default extension to description logics for use in an intelligent search engine, Dwebic. Besides the standard inferences, Lambrix added a new kind of inference to description logic framework to describe whether an individual belongs to a concept from a knowledge base. Calvanese [20] proposed a formal framework to specify the mapping between the global and the local ontologies. Maedche [21] also proposed a framework for managing and integrating multiple distributed ontologies. Stuckenschmidt [6] exploited partial shared ontologies in multi-agent communication using an approximation approach of rewriting concepts. However, default information was not considered in these different frameworks and systems. An important feature of our formal framework distinguished from other work is that our default extension approach is based on DDL. To our best knowledge, little work has been done to pay attention to default extension to DDL for communication among agents.

There is an alternative proposal for dealing with the problem of the example shown in Figure 2.1. For example, if the term SPARROW instead of BIRD in ontology 1 is mapped into the term

NON_SPEAKING_ANIMAL

in ontology 2, and the term PARROT in ontology 1 is not mapped into the term NON_SPEAKING_ANIMAL, then there is no default information to be considered. It seems that we have avoided the problem of default information between the two ontologies using the inter-ontology mapping. However, in fact, this approach is exhausted and unscalable. If there are a lot of terms belonging to the subclasses of BIRD to be added into ontology 1, we have to map every one of these added terms into NON_SPEAKING_ANIMAL in ontology 2. In the situation, we will find the alternative approach is much exhausted and unscalable. In contrast to the alternative approach, our default extension approach to DDL considers the inter-ontology mapping efforts and the scalability of ontologies used by different agents as key features.

Regarding to the complexity issue of the proposed default satisfiability algorithm, we will find that the algorithm increase no more complexity than satisfiability algorithm for $\mathcal{ALCN}$. It means that we can perform reasoning with strict information as well as default information in the same time and space complexity. The future work includes a flexible mechanism for parsing exchanged messages among agents. ACLs are used to construct and parse exchanged messages required by both participants. Then, concepts defined in DAML+OIL ontology language can be readily combined with the mechanism, thus increasing the flexibility of messages, and hence accessibility and interoperability of services within open environments.

7. Conclusion. In this paper, an approach is proposed to enables agents using different ontologies on the Web to exchange semantic information solely relying on internally provided mapping between the ontologies. Because of the semantic heterogeneity among these ontolgies, it is difficult for an agent to understand the terminology of another agent. To get complete and correct semantic information from multiple ontologies used by different agents, default information among these ontologies should be considered. Our approach is based on default extension to DDL. The distributed terminological knowledge base is originally used to present strict information. To perform default reasoning based on DDL, strict as well as default information is taken into account. Then, all of default information above is added into an extended default distributed terminological knowledge base (EDDT), which is constructed from a default distributed terminological knowledge base (DDT). The default Tableau algorithm is used on EDDT where different rules have different priority: exceptional rules have the highest priority, and fulfilled rules the least. Reasoning with default information provides agents using different ontologies with stronger query capability. In our opinion, a query based on DDT can boil down to checking default satisfiability of complex concept in accord with the query.

Our approach enables agents using different ontologies on the Web to exchange semantic information solely relying on internally provided mapping between the ontologies. But so far, our approach is considered as a basic mechanism for facilitating agent communication. To apply it in practice, there is still a lot of work to be done [23]. For example, more sophisticated agent communication protocols, similar to KQML [22] and FIPA [24], have to be developed for getting complete and correct information through agents. Using the communication protocols, concepts defined in DAML+OIL ontology language can be readily combined with the mechanism, thus increasing the flexibility of messages, and hence accessibility and interoperability of services within open environments.

Acknowledgments. This research is partially supported by the National Grand Fundamental Research Program of China under Grant No.TG1999035805, 2002CB312005, the Chinese National “863" High-Tech Program under Grant No.2001AA113010.

REFERENCES

[1] R. T. Payne, et al, *Communicating Agents in Open Multi Agent Systems*, In First GSFC/JPL Workshop on Radical Agent Concepts (WRAC'02), 2002.
[2] V. Damjanovic, et al, *Web Agent's Enclaves—A New Opportunity for the Semantic Web Services*, In Proceedings of WWW2004, 2004.
[3] T. Berners-Lee, et al, *The Semantic Web*, Scientific American, 284(5), (2001), pp. 34–43.
[4] T. Berners-Lee, *The Semantic Web—XML2000*, 2000. http://www.w3.org/2000/Talks/1206-xml2k-tbl/
[5] J. Hendler, *Agents and the Semantic Web*, IEEE Intelligent Systems, 16(2), (2001), pp.30–37.
[6] H. Stuckenschimdt, *Exploiting Partially Shared Ontologies for Multi-Agent Communication*, In Proceedings of CIA'02, 2002.
[7] M. Uschold, Barriers to Effective Agent Communications, In Proceedings of Ontologies in Agent Systems (OAS'01), 2001.
[8] M. Klein, *Combining and relating ontologies: an analysis of problems and solutions*, In Proceedings of IJCAI'01 Workshop: Ontologies and information sharing, Seattle, USA, 2001.
[9] H. Wache, et al, *Ontology-Based Integration of Information-A Survey of Existing Approaches*, In Proceedings of IJCAI'01 Workshop: Ontologies and Information Sharing. USA, (2001) pp. 108–117.
[10] F. Baader, et al, *The Description Logic Handbook: Theory, Implementation and Applications*, Cambridge University Press, Cambridge, 2003.
[11] F. Baader, et al, *Description logics as ontology languages for the semantic web*, Lecture Notes in Artificial Intelligence, Springer-verlag, 2003.
[12] I. Horrocks, et al, *Reviewing the design of DAML+OIL: language for the Semantic Web*, In Proceedings of AAAI'02, Canada, (2002), pp. 792–797.
[13] W3C, *OWL Web Ontology Language Reference*, 2004. http://www.w3.org/TR/owl-ref/
[14] A. Borgida, et al, *Distributed Description Logics: Directed Domain Correspondences in Federated Information Sources*, Journal of Data Semantics, l,1, Springer Verlag, (2003) pp. 153–184.
[15] D. Connolly. et al, *DAML+OIL (March 2001) Reference Description*, 2001. http://www.w3.org/TR/daml+oil-reference.
[16] F. Baader and U. Sattler, *An Overview of Tableau Algorithms for Description Logics*, In Proceeding of Tableaux2000, 2000.
[17] F. Baader and B. Hollunder, *Embedding Defaults into Terminological Representation Systems*, Journal of Automated Reasoning, 14, (1995) pp. 149–180.
[18] U. Straccia, *Default Inheritance Reasoning in Hybrid KL-ONE-style Logics*, In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'93), (1993) pp. 676–681.
[19] P. Lambrix, et al, *A Default Extension to Description Logics for use in an Intelligent Search Engine*, In proceedings of 31st Annual Hawaii International Conference on System Sciences, 5, USA, (1998) pp. 28–35.
[20] D. Calvanese, et al, *A Framework for ontology Integration*, In I. Cruz, S. Decker, J. Euzenat, and D. McGuinness, (eds.), The Emerging Semantic Web _ Selected Papers from the First Semantic Web Working Symposium, IOS Press, (2002) pp. 201–211.
[21] A. Maedche, et al, *Managing multiple and distributed ontologies on the Semantic Web*, In The VLDB Journal-Digital Object Identifier (DOI), 12, (2003) pp. 286–302.
[22] T. Finin, et al, *KQML as an agent communication language*, In Proceeding of the third Conference on Information and Knowledge Management (CIKM'94), ACM press, 1994.
[23] L. Gasser, MAS infrastructure definitions, needs and prospects, In Proceedings of ICMAS2000, 2000.
[24] FIPA, *Foundation for Intelligent Physical Agents*, 2004. http://www.fipa.org/

Edited by: Shahram Rahimi, Raheel Ahmad
Received: October 10, 2005
Accepted: March 19, 2006

Scalable Computing: Practice and Experience
Volume 7, Number 1, pp. 23–36. http://www.scpe.org

ISSN 1895-1767
© 2006 SWPS

A WS-AGREEMENT BASED RESOURCE NEGOTIATION FRAMEWORK FOR MOBILE AGENTS

D. G. A. MOBACH, B. J. OVEREINDER, AND F. M. T. BRAZIER*

Abstract. Mobile agents require access to computing resources on heterogeneous systems across the Internet. They need to be able to negotiate their requirements with the systems on which they wish to be hosted. This paper presents a negotiation infrastructure with which agents acquire time-limited resource contracts through negotiation with one or more mediators instead of individual hosting systems. Mediators represent groups of autonomous hosts. The negotiation protocol and language are based on the WS-Agreement Specification, and have been implemented and tested within the AgentScape framework.

Key words. mobile agents, resource management, agent-based negotiation, WS-Agreements

1. Introduction. One of the assumptions behind the mobile agent paradigm in open, heterogeneous environments is that agents will have access to computing resources. Little thought has been given to the way in which this can be implemented. Not only do they need access, they need to be able to plan coordinated resource usage across multiple domains. Recently, negotiation of the conditions and quality of service of resource access has been considered to be an important capability for distributed, service-oriented architectures. This paper focuses on the negotiation of resource access for mobile agent applications deployed on Internet-scale, open distributed systems. The resources required by agents can vary from CPU type, bandwidth, to the provision of specific services (e. g., databases, web servers, etc.), and level of security required, depending on the task at hand. Well-defined, open protocols and mechanisms are necessary for agents to negotiate their resource access requirements with heterogeneous hosts.

This paper presents a negotiation infrastructure within which individual agents acquire time-limited contracts for the resources they need, through negotiation with one or more system domain coordinators: mediators representing multiple autonomous hosts. The protocols with which agent applications, domain coordinators, and hosts interact, are based on the WS-Agreement Specification [1] with application dependent domain ontologies for specific resources.

The next sections present the negotiation infrastructure, including the model and the architecture. Section 4 describes a specific implementation of this architecture which is integrated within the AgentScape framework. The application dependent domain ontology for specific computer resources is presented together with examples of the WS-Agreement based protocol. In Section 5, two different policies for request distribution by the domain coordinators are compared empirically and evaluated. The paper concludes with related work and discussion.

2. Negotiation infrastructure. The overall goal and use of the negotiation infrastructure is to allow for the negotiation of terms of conditions and quality of service of resource access by agents. The negotiation model includes the exchange of agreement offers and acceptance of the offers between different parties.

2.1. Design Goals. The negotiation infrastructure has to deal with (i) large numbers of heterogeneous agents, and (ii) dynamic groups of heterogeneous hosts each with their own specific sets of requirements.

From the agent's perspective, the negotiation infrastructure defines a uniform and straightforward negotiation protocol and well-defined interface. Agents are not interested in knowing how the process of allocating specific resources to specific hosts is achieved: their interest is to acquire the resources they need. The negotiation infrastructure needs to hide the details from the agent applications.

On the other side, hosts need to keep full control over their own system, over the use of their resources by agent applications. Negotiation policies spanning multiple hosts, allowing specification of resource access and usage policies over a set of hosts (e.g., for load balancing purposes, or virtual organization-wide policies, etc.) must also be facilitated.

2.2. Negotiation Model. In our negotiation model, *hosts* (H) are autonomous entities that provide *resources* (R) to *agents* (A) under specific usage and access policies. Hosts are aggregated into *virtual domains*. The *domain coordinator* (DC), represents the hosts (H) within a virtual domain in the negotiation process, negotiating with both agents and hosts. Figure 2.1 shows an overview of the model.

*IIDS Group, Department of Computer Science, Vrije Universiteit Amsterdam, de Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands

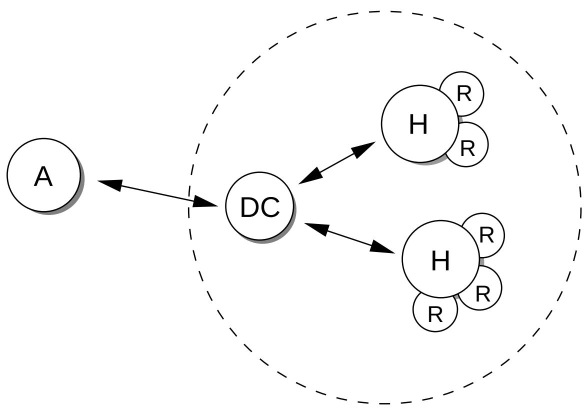


Fig. 2.1. *Negotiation model overview.*

The use of a mediating domain coordinator makes a two-layered negotiation process within the model possible. Agents negotiate resource access with domain coordinators, and domain coordinators, in turn, negotiate with groups of host managers in virtual domains to obtain the actual resources agents require. The results of negotiation are time-limited contracts specifying which resources may be accessed during the time span of the contract, and under which conditions the resources may be used. Agents can negotiate their options with domain coordinators of multiple domains, and select the DC that provides the best offer.

In the model presented in this paper, a domain coordinator represents a virtual organization of resource providers. Agents are unaware of the individual resources behind a domain coordinator: a domain coordinator is viewed by agents to be a single virtual resource provider. The task of selecting one appropriate offer (based on the available resources at a specific point in time) has been delegated to the domain coordinator. Alternatively, a domain coordinator could return a set of possible offers, letting a requesting agent choose the most appropriate. The model presented in this paper supports both options, but only the first is discussed. Section 6 addresses the second option in more detail.

The negotiation protocol and language used in our negotiation model are based upon the *WS-Agreement Specification* [1]. This specification defines the format used to specify agreement descriptions and agreement interactions.[1] The specification defines an XML-based language for agreements between resource providers (hosts) and consumers (agents), and a protocol for establishing these agreements (these agreements are time-limited contracts in our model). Agreement *terms* are used to describe the (levels of) service involved. Two types of terms are distinguished for agreement specifications: (i) *service description terms*, describing the services to be delivered under the agreement, and (ii) *guarantee terms*, expressing the assurances on service quality (e.g., minimum bounds) for the services described in the service description terms. An agreement specification also contains a *context* section, containing meta information about the agreement (see Figure 2.2). This section of the agreement can be used to specify the parties of the agreement, the duration of the agreement, etc. The specification of domain-specific term languages is explicitly left open.

The WS-Agreement interaction model (see Figure 2.3) defines that consumers (C) can request agreements from resource providers (P) by issuing an agreement *request* based on available agreements *templates*, which, if accepted, result in new *agreements.*

In the proposed negotiation model, hosts provide an agreement interface to the domain coordinator. The domain coordinator aggregates the templates offered by the hosts into composed templates. The domain coordinator makes these combined templates available to agents. Agreement requests made by agents are received by the domain coordinator. The domain coordinator negotiates an agreement with the hosts with requested resources.

The interaction protocol as specified in the WS-Agreement Specification only allows for a single "request, accept" interaction, in which the requesting party receives either an *accept* of *reject* message from the providing party as a response to an agreement request. This is a very limited interaction model. In the model proposed in this paper, an additional *accept/reject* interaction sequence is introduced, allowing the requesting party to

[1]This specification is currently under development by the Global Grid Forum's Grid Resource Allocation and Agreement Protocol Working Group.

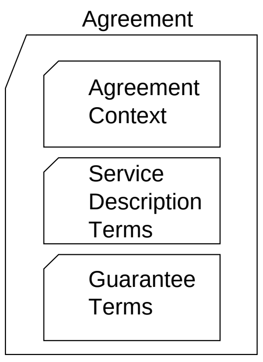


Fig. 2.2. *WS-Agreement contents.*

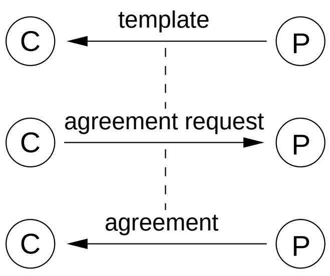


Fig. 2.3. *WS-Agreement protocol.*

explicitly accept or reject an offer created by the providing party. For example, in the context of mobile agent applications, this allows agents to negotiate with multiple domain coordinators simultaneously, and accept the best offer from the set of offers received. Additionally, an explicit *request for templates* interaction is specified. This step in the protocol allows for the initial exchange of information between agents and a domain coordinator, for example for authentication purposes. Figure 2.4 shows the extended interaction model.

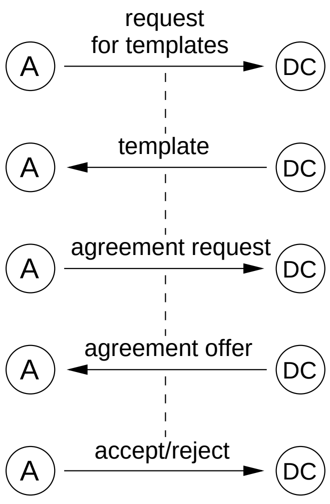


Fig. 2.4. *Extended WS-Agreement protocol.*

3. Negotiation Architecture. The negotiation architecture defines the subsystems and interfaces of the negotiation infrastructure. The two important subsystems host manager and domain coordinator and their interfaces are presented in detail.

3.1. Host Manager. A host manager is responsible for providing and managing resources on its host (see Fig. 2.1). This includes functionality for negotiation, creation, and enforcement of agreements. It is the responsibility of the host manager to translate resource usage and access policies into templates on demand. These templates specify which resources can be made available at a specific point in time. The offer a host makes on request of a domain coordinator is based on these templates. After the negotiation phase, the host manager monitors and controls the resource usage to ensure that agreements are honored.

Figure 3.1 shows the architecture and negotiation interface of a host manager. The agreements in the model are time-limited contracts: agreements that expire after some predetermined time. In the presentation of the architecture, the term *lease* is used instead of time-limited contract. Each host manager is equipped with three modules: a *leasing module*, implementing the main negotiation functionality; a *policy manager* containing *resource policies*, which are applied by the leasing module; a *resource manager* with *resource handlers*, allowing monitoring and control of resource access. The components of the host manager shown in Fig. 3.1 are further described below.

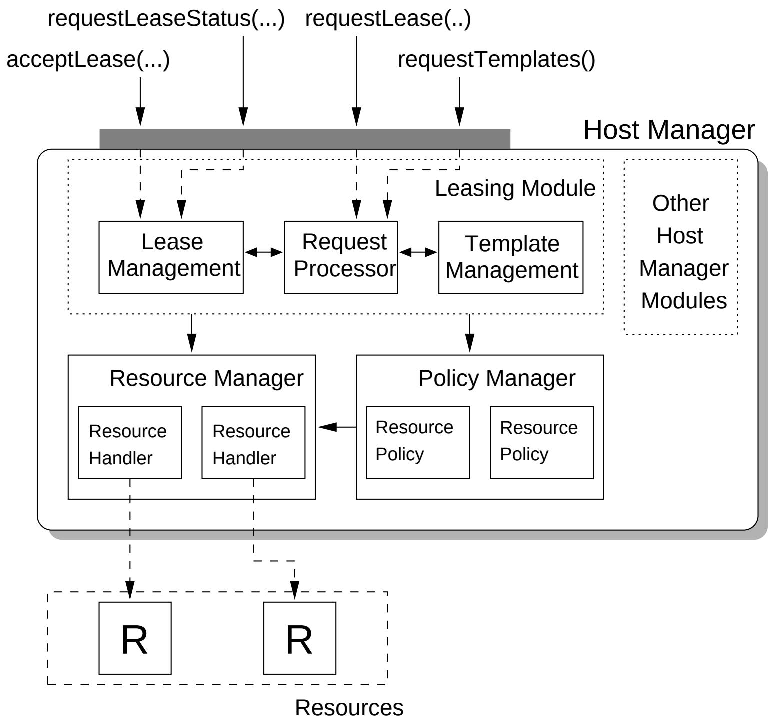


FIG. 3.1. *Components within the Host Manager.*

3.1.1. Leasing Module. The leasing module in the host manager implements the negotiation and agreement protocol. The functionality of the leasing module is available via the interface of the host manager.

Leasing Interface. The leasing interface offered by host managers to their location manager contains the following calls:

- `requestTemplates(): template-list`
  Request the available lease templates.
- `requestLease(LeaseRequest): lease`
  Request a lease based on the supplied lease request.
- `acceptLease(LeaseID)`
  Accept a lease. Returns the accepted lease document.
- `requestLeaseStatus(LeaseID): lease`
  Request the current status of a lease. Returns a lease document, including the current status of each term.

Request Processor.

- Responding to template requests from the domain coordinator according to local policies.

- Creating lease offers. This involves determining the availability of the requested resources, and creating offers based on the incoming request, resource usage and access policies, and the current status of the resources.

Template Management.

- Creating templates based on available resources, resource usage, and access policies, and actively maintaining this information. Note that policies can be dynamic, that is, change over time (e.g., half of available capacity can be reserved during office hours, complete capacity is available outside office hours).

Lease Management.

- Enforcing the accepted leases. This involves ensuring that the resource manager module performs the required resource negotiation tasks.
- Handling expiration of leases. This involves freeing the resources specified in the expired lease, and possibly sending notifications of lease expiration to the domain coordinator.
- Maintaining lease offers: removing the offers after a certain set time, or implementing the offer after notification of acceptance has been received.
- Handling requests for status information on the running leases.
- Handling violation of leases. In cases where resource usage cannot be strictly enforced, and only monitoring can be performed, lease violations should be handled. When an application violates the conditions set in a lease, appropriate actions should be performed, such as suspending or killing the violating agent.

3.1.2. Policy Manager. The policy manager module contains resource policy descriptions which can be used by the leasing module during the processing of requests. Policies can be defined for specific resources, or policies can be defined covering other aspects of incoming requests (identity of the requesting application, or "global" host policies such as the total number of requests, etc.). A resource policy can contain static information, such as the maximum number of allowed requests for a resource, but can also refer to the monitoring capabilities of resource handlers to incorporate up-to-date monitoring data concerning the resources to which the policy applies.

3.1.3. Resource Manager Module. The resource manager module contains a set of resource handlers, enabling the leasing module to manage resources available on the host. Each resource at a host is represented by a resource handler. The handler implements a resource independent interface for the leasing module to monitor and control the resources. Each resource handler supports: (i) creation of resource reservations based on lease offers; (ii) implementation of the reservation, which activates the resource handler to start monitoring resource consumption with respect to accepted leases; (iii) release of a reservation, freeing the resource (amount) related to expired or violated leases. Each resource handler also supports a monitoring interface, allowing for retrieval of resource specific monitoring information, to be used in, for example, resource policies.

- `reserve(LeaseRequest): ReferenceID`
  Can be used to reserve a resource (amount) for a specific lease request. The resource handler inspects the request, and creates a reservation. A reference identifier is returned to enable further management of the reservation.
- `implement(ReferenceID): void`
  Used to request implementation of a reservation (indicated by `ReferenceID`).
- `release(ReferenceID): void`
  Release an implemented resource reservation (indicated by `ReferenceID`).
- `getStatus([ReferenceID]): status`
  Used to request the status of a reservation. Returned value can be one of: *initialized*, *reserved*, *active*, *violated*.
- `getMonitorValue(SensorID): domain_specific_value`
  Used to request resource specific monitoring information concerning a resource.

3.2. Domain coordinator. The domain coordinator abstracts from the individual hosts (resource providers) and presents the aggregated resources as one virtual resource provider. The domain coordinator is responsible for resource access negotiation with applications and its enforcement. To this purpose it provides applications with templates of resources available within its domain at the time requested. The domain coordinator, in turn, requests and receives information on availability of resources from its hosts, and combines this information if, and when appropriate, to construct application directed templates.

Once a template-based request is received from an application, the domain coordinator pursues delegation of resources to hosts. Upon receiving the host bids, the domain coordinator chooses based on available templates, host and domain policies, and returns a proposed lease if possible. If a proposed lease is accepted, the domain coordinator is responsible its effectuation and enforcement.

Figure 3.2 shows an overview of the leasing module within the domain coordinator.

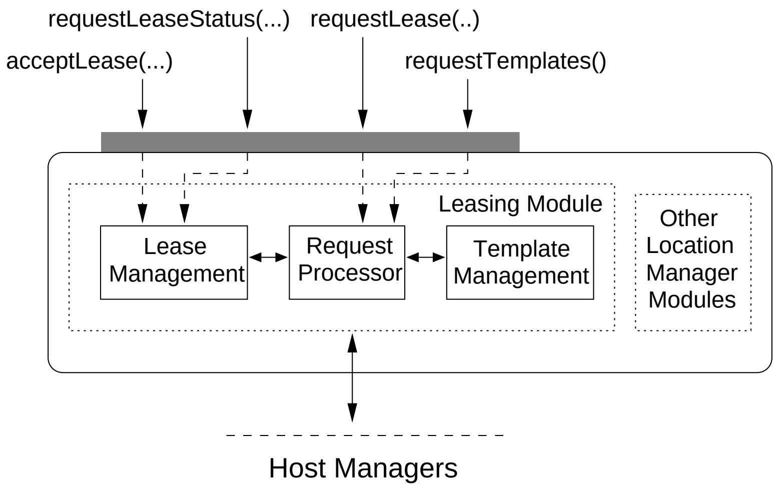


Fig. 3.2. *Leasing components within the domain coordinator.*

Request Processor. This component is responsible for the following tasks:

- Processing requests for templates by applications. This implies checking policies to determine to which template information the application is entitled.
- Processing requests for leases by applications. This involves determining whether the request is based on a valid template, and whether the request exceeds the bounds set by that template.
- Handling lease offers returned by hosts in response to requests. If more than one host was sent the same request, a choice has to be made between their offers. In addition, if the offers are part of a request based upon a combined template, the offers are combined into a single offer for the application. Further, when a lease proposal is accepted by an application, the hosts offering the lease are informed of acceptance.
- Determining from which hosts offers are requested. This involves determining which host(s) are offering relevant templates, and possibly splitting the request into multiple requests for different hosts, if a combined lease template was used by the application.

Template Management. This component requests, creates and maintains information about the templates on which leases are based. This component performs the following tasks:

- Obtaining and maintaining template information of the hosts currently in the domain.
- Creating template combinations of resources from multiple hosts in a single template. This involves applying local template policies specifying which host templates can or cannot be combined.

Lease Management. The lease management component maintains information about leases, lease requests made by applications, and lease proposals from hosts, and performs the following tasks:

- Maintaining status information of current valid leases. This involves actively or passively retrieving lease status information from the hosts responsible for enforcing the leases acting appropriately upon lease expiration.
- Maintaining information of currently outstanding lease proposals.

4. AgentScape Negotiation Architecture. The negotiation architecture described above has been implemented in the AgentScape framework, a framework for heterogeneous, mobile agents. This section describes how the subsystems have been instantiated, and provides examples of how the agreement-based negotiation is used to create leases for agent applications using the AgentScape middleware.

4.1. AgentScape. The AgentScape middleware [8] consists of two layers. At the base of the middleware is the *kernel*, offering low-level secure communication between middleware processes, and facilities for secure agent mobility. On top of the AgentScape kernel, middleware processes provide higher-level middleware functionality to agents. For example, *agent servers* provide a run-time environment for agents, and a *Web service gateway* provides agents the ability to communicate with web services using the SOAP/XML protocol. In AgentScape, virtual domains are called *locations*. An AgentScape location consists of one or more hosts running the AgentScape middleware, typically within a single administrative domain.

In addition to the middleware processes described above, each host has a *host manager* middleware process. This process is responsible for managing the middleware components running on the host, and implementing the required negotiation functionality as described in the architecture. Furthermore, each AgentScape location runs a *location manager* process on one of the hosts, which implements management functionality required for managing AgentScape hosts, and which implements the functionality of the domain coordinator, enabling agent application to enter into resource negotiations with locations. Figure 4.1 shows an overview of an AgentScape location.

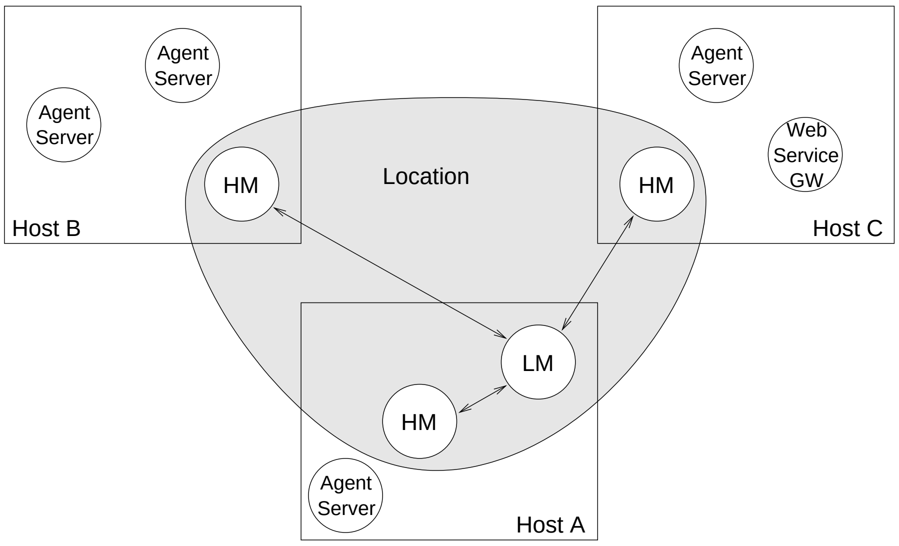


Fig. 4.1. *Overview of an AgentScape location.*

4.2. AgentScape Negotiation Architecture. Within AgentScape, agents can start negotiations with a number of locations, and given the offers the locations provide, select the location offering the best options. The agent then migrates to the location with which agreement has been reached.

4.2.1. AgentScape resources. The AgentScape negotiation architecture defines a set of resources that can be allocated and used by agents in the AgentScape specific ontology. This ontology is used during negotiation. Currently, the following resources are included in this ontology:

- CPU time: The time (in milliseconds) that an agent spends on an agent server.
- Communication bandwidth: The number of bytes/second that an agent may send to other agents.
- Memory: The amount of RAM an agent may consume while running on an agent server.
- Web service access: The web services that an agent is allowed to access using the AgentScape Web Service Gateway.
- Web service call rate: The number of calls that an agent is allowed to do on a web service using the gateway.
- Disk space: The amount of disk space an agent is allowed to use while running on an agent server.

Additional resources can be defined in the future, as the functionality offered by AgentScape is extended. The resources are specified in the XML Schema language, enabling the use of these definitions within the agreement-based negotiation sequence. As an example, consider the three resources specified in Example 4.1. In this example, the `time-on-cpu` resource and the `communication-bandwidth` resource are defined as simple integer values representing the number of milliseconds and the number of Kilobytes/second respectively. The `web-service-access` resource is defined as a list of service names (strings) representing the list of services which may be accessed.

```
<xsd:simpleType name="time-on-cpu"
                type="xsd:positiveInteger" />

<xsd:simpleType name="communication-bandwidth"
                type="xsd:positiveInteger" />

<xsd:complexType name="web-service-access">
  <xsd:all>
    <xsd:element name="service-name" type="xsd:string"
                 minOccurs="1" maxOccurs="unbounded"/>
  </xsd:all>
</xsd:complexType>
```

Example 4.1
AgentScape resource definitions.

The AgentScape specific language is used within the lease model to express resource requirements and usage conditions. In Example 4.2, an example of agent resource requirements is shown. In this example, an agent requests 50 seconds of CPU time, and 50 Kb/s of communication bandwidth.

```
<!-- requirement: 50 seconds CPU time -->
<agentscape:time-on-cpu>
  50000
</agentscape:time-on-cpu>

<!-- requirement: 50Kb/s bandwidth -->
<agentscape:communication-bandwidth>
  51200
</agentscape:communication-bandwith>
```

Example 4.2
Agent resource requirements.

4.3. AgentScape Host Manager. The AgentScape host manager is responsible for offering resources to the location manager. Based on its own information on the status of its resources, and its own policies regarding these resources, the host manager creates a set of templates. Example 4.3 shows an example of a template, using the syntax as defined in the WS-Agreement Specification. The template specifies that this host can now offer two resources, each with specific access conditions. For the first resource: the `time-on-cpu` resource, a maximum value of 100 seconds is specified. The second resource, `communication-bandwidth`, is not restricted by the template.

4.4. Location Manager. The location manager enters into negotiation with host managers within its location on behalf of agents. The location manager maintains information on the templates offers by each of the hosts within the location, and uses this information to provide templates to agents. Agents base their requests for leases to the location manager on these templates. As an example, consider the following request, in which an agent requests a location for 50 seconds of CPU time, and 50 Kb/s of communication bandwidth.

To meet lease requests by agents, the location manager enters into negotiation with the relevant hosts in its location (those that can provide the resources requested). For each request received from an agent, one or more suitable hosts are selected (based on their templates). Each of the hosts then creates an offer based on the current resource conditions. The location manager selects one of the offers, and discards the others, or combines a number of offers into a composed offer. The selected offer is returned to the agent. As mentioned in Section 2.2, multiple offers can be returned to the agent, but does not comply with the AgentScape model.

In the following example, a location manager has received a request from an agent, and has selected two hosts within its location to which it forwards the request. The hosts determine if and to which extent the request can be fulfilled, and return their offers (proposed leases) to the location manager. In Example 4.5, Host 1 returns a proposal in which the requested CPU-time is unchanged with respect to the request from

```
<wsag:Template>
  <wsag:Name>Template1</wsag:Name>
  <wsag:Context/>
  <wsag:Terms/>
  <wsag:CreationConstraints>
    <wsag:Item>
      <wsag:Location>//wsag:ServiceDescriptionTerm//
         agentscape:time-on-cpu
      </wsag:Location>
      <xs:maxInclusive xs:value="100000">
    </wsag:Item>
    </wsag:Item>
      <wsag:Location>//wsag:ServiceDescriptionTerm//
            agentscape:communication-bandwidth
      </wsag:Location>
    </wsag:Item>
  </wsag:CreationConstraints>
</wsag:Template>
```

Example 4.3
AgentScape resource template.

```
<wsag:AgreementOffer>
  <wsag:Name>Offer1</wsag:name>
  <wsag:Context>
    <wsag:AgreementInitiator>
      agentX
    </wsag:AgreementInitiator>
    <wsag:TemplateName>
      Template1
    </wsag:TemplateName>
  </wsag:Context>
  <wsag:Terms>
  <wsag:All>
    <wsag:ServiceDescriptionTerm
          wsag:Name="TimeOnCPU"
          wsag:ServiceName="LocationY">
      <agentscape:time-on-cpu>
        50000
      </agentscape:time-on-cpu>
    </wsag:ServiceDescriptionTerm>
    <wsag:ServiceDescriptionTerm
          wsag:Name="Communication"
          wsag:ServiceName="LocationY">
      <agentscape:communication-bandwidth>
        51200
      </agentscape:communication-bandwidth>
    </wsag:ServiceDescriptionTerm>
  </wsag:All>
  </wsag:Terms>
</wsag:AgreementOffer>
```

Example 4.4
Lease request made by agent.

the agent, and communication-bandwidth is decreased to 10 Kb/s. Host 2 also returns a proposal in which the requested `time-on-cpu` is reduced to 40 seconds, and `communication-bandwidth` is decreased to 30 Kb/s. Also, an `ExpirationTime` element is added to the context section of the proposal, indicating when the lease will expire, if accepted by the agent. Host 1 defines an expiration time of 23:04:44 upon which it no longer guarantees the requested resources, and Host 2 defines an expiration time of 23:10:00.

The proposals are received and compared by the location manager. Host 1 offers fully the requested `time-on-cpu`, but offers a `communication-bandwidth` which is substantially lower than the requested bandwidth. The offer made by Host 2 offers a lower `time-on-cpu` value, but does offer a bandwidth value which is

```
<wsag:Agreement>
  <wsag:Context>
    <wsag:AgreementInitiator>
      AgentX
    </wsag:AgreementInitiator>
    <wsag:AgreementProvider>
      Host1
    </wsag:AgreementProvider>
    <wsag:ExpirationTime>
      2005-07-23T23:04:00
    </wsag:ExpirationTime>
  </wsag:Context>
  <wsag:Terms>
    <wsag:All>
      <wsag:ServiceDescriptionTerm
            wsag:Name="TimeOnCPU"
            wsag:ServiceName="LocationY">
        <agentscape:time-on-cpu>
          50000
        </agentscape:time-on-cpu>
      </wsag:ServiceDescriptionTerm>
      <wsag:ServiceDescriptionTerm
            wsag:Name="Communication"
            wsag:ServiceName="LocationY">
        <agentscape:communication-bandwidth>
          10240
        </agentscape:communication-bandwidth>
      </wsag:ServiceDescriptionTerm>
    </wsag:All>
  </wsag:Terms>
</wsag:Agreement>
```

```
<wsag:Agreement>
  <wsag:Context>
    <wsag:AgreementInitiator>
      AgentX
    </wsag:AgreementInitiator>
    <wsag:AgreementProvider>
      Host2
    </wsag:AgreementProvider>
    <wsag:ExpirationTime>
      2005-07-23T23:10:00
    </wsag:ExpirationTime>
  </wsag:Context>
  <wsag:Terms>
    <wsag:All>
      <wsag:ServiceDescriptionTerm
            wsag:Name="TimeOnCPU"
            wsag:ServiceName="LocationY">
        <agentscape:time-on-cpu>
          40000
        </agentscape:time-on-cpu>
      </wsag:ServiceDescriptionTerm>
      <wsag:ServiceDescriptionTerm
            wsag:Name="Communication"
            wsag:ServiceName="LocationY">
        <agentscape:communication-bandwidth>
          30720
        </agentscape:communication-bandwidth>
      </wsag:ServiceDescriptionTerm>
    </wsag:All>
  </wsag:Terms>
</wsag:Agreement>
```

Example 4.5
Host lease proposals.

closer to the requested value than the offer of Host 1. The location manager makes a selection between these offers based on current selection policies, and communicates this offer to the agent. In our example, the location manager chooses the proposal made by Host 2. The agent chooses to accept the offer. After acceptance, the agent has a limited time in which it must migrate to the target location, or the lease offer will expire. After the arrival of the agent at the target location, the agent is allowed to consume the agreed upon resources until the lease expires.

```
sendMessage(agentID, messageContent)
receiveMessage()
move(LocationID)
kill()
suspend(timeOut)
requestTemplates(LocationID)
requestLease(LocationID, leaseRequest)
requestLeaseStatus(LocationID, leaseID)
acceptLease(LocationID, leaseID)
requestWSDLAccess(...)
sendSOAPRequest(...)
...
```

Fig. 4.2. *Lease related calls on the AgentScape agent interface.*

4.5. AgentScape Agent Interface. The interface presented to agents by the AgentScape middleware contains several lease related calls, as shown in Figure 4.2. These calls enable agents to enter into resource lease negotiations with AgentScape locations.

5. Experiments. To evaluate the implementation and assess the operation of the negotiation architecture described above, several experiments have been performed. The first set of experiments centered on the ability of the negotiation architecture to accommodate domain-wide resource policies. The second set of experiments focused on the use of the negotiation architecture to apply "quality of service" policies using individualized host policies.

5.1. Experimental setup. A distributed AgentScape location is set up consisting of nine hosts. Eight hosts are configured to run a host manager and an agent server, and one host is configured to run a location manager. The location manager implements the domain coordinator negotiation functionality. In each of the experiments, agents migrate to the location after a lease has been acquired through negotiation with the location manager. The hosts used for the AgentScape location are part of the DAS-2 cluster at the Vrije Universiteit Amsterdam, consisting of Dual Pentium-III nodes connected by Fast Ethernet (Myrinet-2000 is available between machines at each cluster, but was not used in these experiments). The agents are inserted from a host outside the DAS-2 cluster, also connected by Fast Ethernet.

In the experiments, CPU-time is the main subject of the negotiation process. In each experiment, one thousand agents are inserted into the location. For each agent, a "desired" CPU-time amount is generated according to the Weibull distribution (scale = 3.0, shape = 2.0, mean = 26.587 seconds). This value from the distribution is then used to create a lease request which is then sent to the location. The intervals between lease requests of individual agents are distributed according to the Poisson distribution (mean = 2 seconds). Each lease request received by the location manager is translated into lease requests to the 8 host managers within the location. Each host manager then responds with a lease offer if the requested value is in line with the local CPU-time policy, or responds with an empty offer if the requested value is not in line with the policy. In the experiments, the load on a host is represented as the number of agents running on a host, measured at one second intervals.

5.2. Domain-wide negotiation policy experiments. In the area of distributed systems it is useful to apply domain policies facilitating the distribution of computational load across available hosts in the environment. Two straightforward types of policies are based on the principles of: (1) time-division, in which computational load is scheduled for execution at different times, and (2) space-division, in which computational load is scheduled on different hosts. In these experiments, a round-robin (space-division) negotiation policy is applied, i. e., a location manager collects offers made by the hosts, and applies a round-robin load balancing policy to select one of the offers made by the hosts. This offer is then sent back as an answer to the original lease request. After acceptance of the lease, an agent is inserted at the host that has been selected during negotiation. The agent will then start to consume CPU-time by performing predefined calculations. When the CPU-time delegated to the agent in the lease is consumed, the agent is stopped and removed from the host. In this experiment, hosts are configured with a negotiation policy dictating that all lease requests should be accepted, regardless of the requested CPU-time value. The location manager selects host manager offers according to a round-robin policy, with the aim of to distribute all agents evenly throughout the location.

As a measure for the balance of the load within the AgentScape location, the "Load Balance Metric" is used, as described by Bunt and Eager [4]. This metric is defined by taking the weighted average of peak-to-mean server load ratios. This ensures that a larger imbalance during high-load situations has a greater effect on the LBM measure than a smaller imbalance during lower-load conditions. The value of the LBM measure ranges from the number of servers (8 hosts in the experiments) to 1, where a lower value represents a higher balance (LBM value 1 means perfect load balance). In Fig. 5.1, the LBM values are graphed, calculated over 10 second intervals. The figure shows that a consistent balance is achieved within the location using the round-robin policy, during the insertion of agents as described in the experimental setup. At the end of the experiment, load balance can no longer be enforced, as all agents have been inserted and load imbalance is induced by the completion of agents at a host, while a fraction of the hosts is still executing long running agents. This is shown in the graph by the sharp increase of the LBM value.

5.3. Differentiated host policy experiments. In the second set of experiments, negotiation policies were applied to implement a quality of service policy aimed at improving responsiveness for agents with a relatively short running time (below the mean value as described above). In the experiments, two different host policies are used: a policy allowing only requests below the mean CPU-time value, and a policy allowing only requests above the mean CPU-time value. (The CPU-time values are taken from the same Weibull distribution as described in Section 5.1.) In each experiment, the number of hosts accepting below-mean and above-mean is varied. The round-robin policy of the location manager is still applied, but within the two host groups

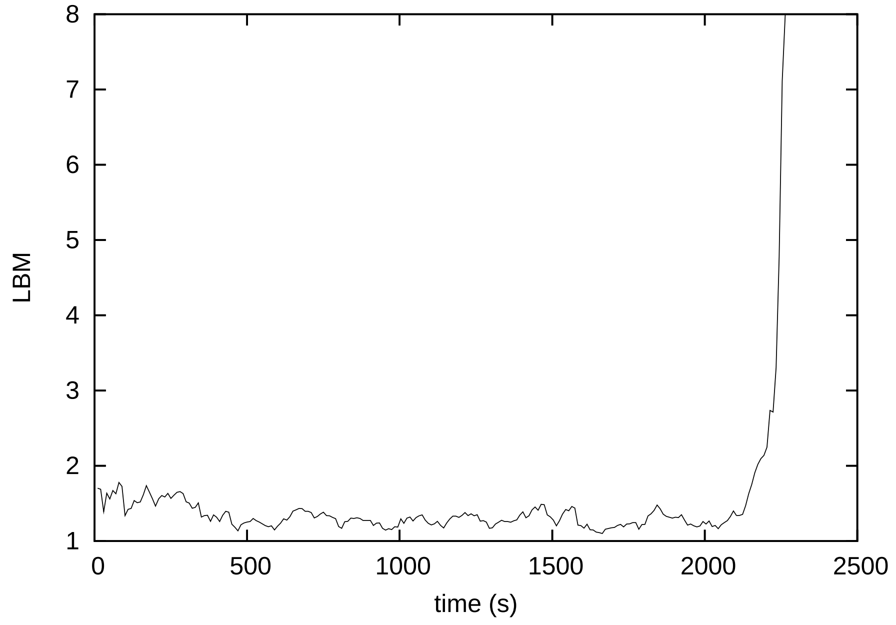


Fig. 5.1. *LBM over 10 second intervals using round-robin negotiation policy.*

separately, as attaining a balanced load within groups is still desirable, but is not feasible across the different groups.

In Table 5.1, the results of these experiments are shown. In the first column, the number of hosts accepting only agents with a CPU-time value below the mean is given. The second and third column present a quality of service percentage for agents with a below-mean and above-mean CPU-time value respectively. The quality of service percentage metric is defined as the actual CPU-time agents have consumed divided by the "wall clock" time agents have spent on a host. The results in the table are the mean over three experiments. A high quality of service percentage of 100 % indicates a perfect quality of service where the resource is completely available to the agent (the agents in the experiments are CPU bound, and, e.g., not waiting for I/O or network communication). A low quality of service percentage means that the agent has to compete with other agents (or generally tasks) to access the resources.

The values in the bottom row are obtained from the load balancing experiments presented in the previous section, in which no differentiation was made based on CPU-time values, and agents could be placed on all hosts. This can be seen as a "reference" value, indicating the responsiveness in the undifferentiated case. From the results it can be argued that a configuration with 8 hosts, where 3 hosts accepting only agents with below-mean CPU-time values (and consequently 5 hosts accepting only above-mean CPU-time), gives agents with a shorter running time a better responsiveness, at a not too great expense for the longer running agents. For 4 hosts reserved for short running agents, the responsiveness dramatically improves with about a factor of 5 compared to the reference results, while the long running agents experience an increased turnaround time of a factor of 1.7.

The experiments have shown that different policies can be relatively easily enforced, both on aggregate location level, enforcing a round-robin load balancing policy, as well as on individual host level, accepting either short or long running agents. It should be stressed that the experiments are not intended to show the performance of specific policies, but rather show how different policies defined on location and host level can be defined and enforced by the resource negotiation infrastructure presented in this paper.

6. Related Work and Discussion. The negotiation architecture described above hides the complexity of managing access and usage of heterogeneous and distributed resources from agents, by providing a uniform negotiation infrastructure aggregating the resources within a virtual domain. The architecture uses the WS-Agreement emerging Grid standard as a basis for its negotiation protocol and language.

The WS-Agreement framework offers an extensible basis for resource management involving distributed heterogeneous resources and distributed applications. In its current state however, the WS-Agreement frame-

# below mean hosts	avg. for below mean agents %	avg. for above mean agents %
2	8.3	38.6
3	24.9	13.2
4	76.3	9.7
5	87.7	5.8
6	90.9	4.5
reference	14.5	16.2

Table 5.1
Quality of service percentage results of the CPU-time differentiated host policy experiments.

work has a number of shortcomings. First, the specification only provides for a basic negotiation protocol and related information structures. This could be sufficient for use in service-oriented environments for which the model is intended, however, in a self-managing application domain, as described in this paper, more elaborate negotiation facilities could provide these applications with more control over allocation and use of resources. Second, the framework does not provide a model describing how enforcement of agreements is to be integrated in the system providing the resources. Although it can be argued that much of this is very domain-specific and cannot be captured in a useful model, the framework could present an abstract model of the required information structures and design of an agreement-based infrastructure supporting the WS-Agreement framework.

In this paper, an extension of the WS-Agreement negotiation protocol is proposed. The addition of an explicit accept/reject interaction sequence allows agents to enter into negotiations with multiple providers and compare received offers. The proposed framework is implemented in the AgentScape middleware. In a recent paper, Paurobally and Jennings [9] also recognize the need for more complex negotiation patterns other than possible within the WS-Agreement Specification. In their paper, richer message types (i. e., *inform* and *bid*) and interaction protocols are proposed in the form of an additional layer, allowing for the specification of agent interaction protocols on top of the WS-Agreement messaging layer. The Grid Resource Allocation Agreement Protocol (GRAAP) working group also extended their work on WS-Agreement with the WS-Agreement Negotiation Specification [2]. Here, a negotiation layer is defined to be incorporated on top of the WS-Agreement Specification. The negotiation layer allows to express negotiation offers in terms expressed in the meta-language already defined in WS-Agreement.

Independent from the WS-Agreement Specification activities, Hung *et al.* [6] proposed a Web service negotiation model called WS-Negotiation. Also, a service level agreement (SLA) template model is presented, with different domain specific vocabularies for supporting different types of negotiation. The negotiation protocol in their model is geared toward integrative negotiation, where both parties locate and adopt the option that provide greater joint utility to the parties taken collectively. The message types reflect this negotiation model and is more extended than the models presented by Paurobally and Jennings [9] and the GRAAP working group [2].

IBM's Cremona [7] (Creation and Monitoring of Agreements) is an effort to create an architecture and set of libraries that implement the WS-Agreement interfaces and agreement (template) management, and provide agreement functionality suitable for implementations in domain-specific environments. The Cremona architecture specifies domain-independent and domain-specific components required for agreement-based management, and the Cremona libraries provide implementations of the agreement interfaces, domain-independent components, and well-defined interfaces for the domain-specific components. Cremona is currently being offered as a part of IBM's Emerging Technologies Toolkit.

The design goals and the realization of the WS-Agreement-based negotiation infrastructure presented in this paper and the Cremona architecture are quite similar. However, the WS-Agreement-based negotiation infrastructure extends the Cremona architecture with the option to combine templates and agreements from multiple resources. The combination of templates and agreements is necessary to accomplish resource aggregation, for example, to implement virtual organizations where multiple resource cooperate to provide a (number of) services.

The concept of leasing has been used in the area of distributed application frameworks, for example in Jini [10], where leases are used for distributed garbage collection. In the Jini framework, clients lease resource access, such as for example service registration within a lookup service. The acquired lease allows a client to make of use of that resource for a limited time-period. When a lease expires, and no explicit renewal is requested by the client (for example because of network failure), the associated resource is made available for other clients, preventing unnecessary resource allocation. This characteristic has been included in the negotiation model presented in this paper. The Jini specification, however, does not cover a negotiation model or protocol specification. In the SHARP [5] architecture, tickets (soft resource claims) can be redeemed by resource consumers for leases (hard resource claims), which guarantee access to a resource. Ticket holders can delegate resources to other principals by issuing new tickets. The goals of the SHARP architecture and the AgentScape negotiation architecture are similar in nature, with the AgentScape negotiation architecture being more oriented towards agent applications.

The focus of our current and future work includes extending the architecture and model with agent level components, allowing application developers to more easily integrate and implement resource negotiation interactions into their applications. As an example, for the AgentScape middleware, a WS-Agreement based Agent Communication Language would enable agents to more easily communicate with the resource negotiation infrastructure. Furthermore, the addition of more expressive and flexible negotiation protocols would allow both applications and resources more fine-grained control of the negotiation process.

As stated in Section 2.2, the current implementation of the domain coordinator in the negotiation infrastructure returns one offer in reply to an agent request. This is an implementation decision and not a limitation of the negotiation model or protocol. If the domain coordinator returns multiple offers, the requesting agent can decide which offer is most appropriate to complete its current task, e. g., considering expected computing time, execution costs, security level, or other uses of resources. Part of the extended negotiation protocol can be the specification by the agent whether it opts for a light-weight negotiation protocol with single offers, or a more complex negotiation protocol with multiple offers.

The negotiation architecture makes it also possible for a virtual provider to check an agent's credentials before even starting to negotiate with an agent. As identity management is an important aspect in the design of large-scale open agent systems [3], this aspect is currently being further explored, in particular in relation to legal implications of the use of mobile agents.

REFERENCES

[1] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, *Web services agreement specification (WS-Agreement) (draft)* 2006, `https://forge.gridforum.org/projects/graap-wg`

[2] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, *Web services agreement negotiation specification (WS-AgreementNegotiation) (draft)* 2004, `https://forge.gridforum.org/projects/graap-wg`

[3] F. Brazier, A. Oskamp, J. Prins, M. Schellekens, and N. Wijngaards, *Anonymity and software agents: An interdiscplinary challenge*, AI & Law, 1-2 (2004), pp. 137–157.

[4] R. B. Bunt, D. L. Eager, G. M. Oster, and C. L. Williamson, *Achieving load balance and effective caching in clustered Web servers*, in Proceedings of the 4th International Web Caching Workshop, San Diego, CA, Apr. 1999, pp. 159–169.

[5] Y. Fu, J. Chase, B. Chun, S. Schwab, and A. Vahdat, *SHARP: An architecture for secure resource peering*, in Proceedings of the 19th ACM Symposium on Operating Systems Principles, Bolton Landing, NY, Oct. 2003, pp. 133–148.

[6] P. C. K. Hung, H. Li, and J.-J. Jeng, *WS-Negotiation: An overview of research issues*, in Proceedings of the 37th Hawaii International Conference on System Sciences (HICSS'04), Big Island, Hawaii, Jan. 2004, pp. 33–42.

[7] H. Ludwig, A. Dan, and R. Keaney, *Cremona: An architecture and library for creation and monitoring of WS-Agreements*, tech. report, IBM Research Division, June 2004.

[8] B. J. Overeinder and F. M. T. Brazier, *Scalable middleware environment for agent-based Internet applications*, in Proceedings of the Workshop on State-of-the-Art in Scientific Computing (PARA'04), Copenhagen, Denmark, June 2004, pp. 675–679. Published in Applied Parallel Computing, LNCS 3732, Springer, Berlin, 2006.

[9] S. Paurobally and N. R. Jennings, *Developing agent Web service agreements*, in Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology, Compiegne, France, Sept. 2005, pp. 464–470.

[10] J. Waldo, *The Jini architecture for network-centric computing*, Communications of the ACM, 42 (1999), pp. 76–82.

Edited by: Shahram Rahimi, Raheel Ahmad
Received: November 8, 2005
Accepted: March 19, 2006

Scalable Computing: Practice and Experience
Volume 7, Number 1, pp. 37–47. http://www.scpe.org

ISSN 1895-1767
© 2006 SWPS

AGENT COMPOSITION VIA ROLE-BASED INFRASTRUCTURES

GIACOMO CABRI*

Abstract. Software agents represent an interesting paradigm to approach complex and distributed systems. Their sociality enables to build multiagent systems, where different agents interact to pursue their goals. Multiagent systems can involve both cooperative and competitive agents. In both cases, the composition of different agents is an issue that must be faced by developers. In this paper, we propose to build infrastructures based on roles, which are abstractions that enable the composition of different agents in an open scenario. Some concrete examples are provided to support our proposal.

Key words. agents, roles, multiagent systems, interaction

1. Introduction. With no doubt software agents have been proposing as an innovative paradigm for some years, and we envision a digital world populated by them to support users belonging to the human world. In fact, they are able to perform tasks on behalf of users, due to their main features—*autonomy, proactiveness, reactivity* and *sociality* [19]. In addition, complex applications can be divided into smaller and simpler tasks, each one delegated to one agent [18]. This leads to systems composed of several agents, called *multiagent systems*, where agents interact and coordinate to carry out a common goal. Going further, in a wide open scenario the social behaviour of the agents implies interactions not only between agents cooperating in one application, but also between agents of different applications, which may have a competitive behaviour, to gain the use of resources [12]. The feature of *mobility* [20], enhancing the autonomy of agents, implies further advantages. Generally, mobile agents can save bandwidth by moving locally to the environments where the resources are located, and do not rely on continuous network connections. Users are not required to be connected to the network continuously: they can send their agents, disconnect, and then reconnect when the agents have carried out their tasks to retrieve them.

However, building multiagent systems is not so easy, because they require a careful and effective composition of different agents, to carry out their tasks; the presence of mobile agents may make the scenario even more complex, since agents can enter and exit an application context dynamically. Composition means not only specifying which agents take part in a given application context, but also taking into account the rules for the interactions between agents and between agents and environments. Developers must face these issues during the entire development process. This paper proposes to compose agents by means of *infrastructures* based on the concept of *role*. A role can be defined as a stereotype of behaviour, and is exploited in particular in a group or organization of entities, each one exhibiting a specific behaviour inside the group. Real life proposes lots of examples where people play roles and the concept of role has been exploited in the Object-Oriented field to design complex applications, in the agent area, and in other proposals related to computer science in general. We show that roles can be exploited to build infrastructures that are useful abstractions to compose different agents in a multiagent system. Such abstractions can then be implemented exploiting a role-based system.

The paper is organized as follows. Section 2 introduces the concept of role for the agents. Section 3 presents the definition of infrastructures based on roles. Section 4 introduce the RoleX system, which can be exploited for a possible implementation. Section 5 shows some examples, giving more details about the implementation. Section 6 reports some related work. Finally, Section 7 concludes the paper and sketches some future work.

2. Roles. One of the most important features of the agents is *sociality*, thanks which complex applications can be built on the base of the interactions between autonomous components (the agents themselves). In this context, it is important that application developers face the engineering of the interactions between agents with appropriate models and tools. To this purpose, a concept that seems to be suitable is *role*. The Oxford Dictionary reports: "*Role: noun 1. an actor's part in a play, film, etc. 2. a person's or thing's function in a particular situation.*" [23]. In describing patterns, Fowler says that roles are "some common behaviour of entities" that "do not have the same behaviour" [17], and points out that the isolation of such common behaviour can simplify the design of applications. Roles have been already exploited in Object Oriented approaches, where a role is applied to an object in order to change its capabilities and behaviour. Other approaches promote roles

*Dipartimento di Ingegneria dell'Informazione, Universita' di Modena e Reggio Emilia, via Vignolese, 905, 41100 Modena, Italy (cabri.giacomo@unimore.it).

as views of a particular object or entity [2], stressing the similarity between roles in computer programs and those in the real life.

Roles have been applied to agents promoting the reuse of solutions and making the definition of interaction scenarios easier. Roles allow not only the agent developers/designers to model the execution environment, but also allow agents to actively "feel" the environment itself. In other words, roles allow the developer and its agents to percept in the same way the execution environment. Roles can also be exploited to model and manage agent interactions [9], embedding all the capabilities (e.g., protocols, events) needed to handle a specific interaction. This is probably the most important meaning of roles: they allow and enable specific interactions, as well as social roles enable people, in the real world, to act in a certain way depending on the role they are playing in the society. There are some characteristics of roles that distinguish them from the concept of agent. The role is *temporary*, since an agent may play it in a well-defined period of time or in a well-defined context. Roles are *generic*, in the sense that they are not tightly bound to a specific application, but they express general properties that can be used in different applications. Finally, roles are *related to contexts*, which means that each environment can impose its own rules and can grant some local capabilities.

To better explain these concepts, we exploit an example in the software auction field, which will be taken up again later. Let us consider a software agent that is interested in acquiring a good or a service in a distributed environment. At runtime, after finding the good/service, it could discover that the good/service is put on sale by a specific auction house. To carry out its task, it can dynamically assume the role of bidder to attend the auction. In this case, the bidder is a behaviour that can be exhibited by different agents, of different applications and with different interests. But the features (or mechanisms) for bidding in a given auction house are independent of agents and can be embedded in the role of bidder. Moreover, different auction houses can provide different mechanisms and policies to rule the interactions. The importance of the use of roles is supported by the fact that they are adopted in different areas of the computing systems, in particular to obtain uncoupling at different levels. Some examples of areas are *security*, in which we can recall the Role Based Access Control (RBAC) [24] that allows uncoupling between users and permissions, and the *Computer Supported Cooperative Work* (CSCW) [29], where roles grant adaptation and separation of duties. Also in the area of software development we can find approaches based on roles, especially in the *object-oriented programming* [13, 22], in *design patterns* [17], and in computer-to-human interfaces [27], which remark the advantages of role-based approaches.

Finally, it is important to note that roles can be exploited in a static or dynamic way. The static way imposes that roles are joined to an agent before its execution, while the dynamic way allows roles to be joined during at run-time, during the application execution.

3. Composing Agents via Role-based Infrastructures. In the introduction, we state that multiagent systems are useful but must be developed carefully; in particular, agents of multiagent systems must be composed and their interactions must be designed. To support this job, we propose to build *infrastructures* as abstractions to compose different agents. These infrastructures are based on roles, and the key idea is that a set of roles determines an infrastructure that specifies which agents (or, better, which roles) take part in an application context and rules the corresponding interactions with other agents and with the environment. Such an infrastructure can be considered also as intermediate between applications and environments; in the following we consider the resources as part of the infrastructures, but actually this model can be adopted even in case of legacy software that remains outside the infrastructure. It is useful to identify some issues inside the infrastructures to simplify the design and to help the implementation and the deployment. We propose to model role-based infrastructures distinguishing three levels (see Figure 3.1):

- the *role level* contains the roles that agents can assume in the environment; each environment has its own set of admitted roles;
- the *policy & mechanism level* aims at defining the policies local to the environment and the mechanisms that implements the interaction among roles;
- finally, the *resource level* contains the local resources, such as information and services.

Above the role level we can find application agents that play roles. In this paper we disregard the internal constitution of agents, focusing on their external behaviours–captured by roles. We assume that agents are *network-aware*, which means that they perceive the network not as a whole flat environment, but instead as a set of environments, each one with given resources and services [8].

The role level can be considered as the interface of an environment for agents. To define the infrastructure, an administrator has to perform the following steps:

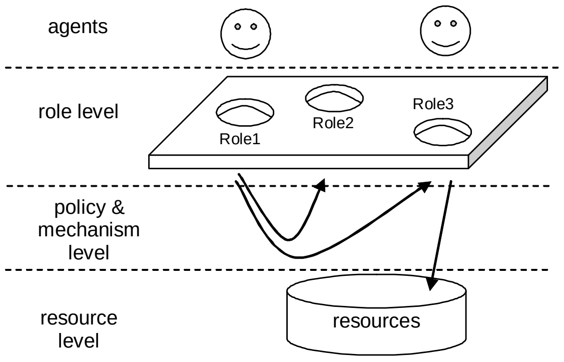


Fig. 3.1. *A role-based infrastructure*

1. to choose the *roles* her/his environment is going to support; often, this choice is implicit in the environment, as shown in the first example of the next section;
2. to define the *policies* by which the chosen roles can interact with each other or with the local resources.

This model of infrastructures leads to advantages at several phases of the application life cycle. At the *design stage*, roles permit separation of concerns, which allows the designer to concentrate on the single (interaction) issue, and the reuse of solutions. At the *development stage*, the reuse of roles permits to avoid the implementation of common (already implemented by someone else) functionalities. At the *runtime*, more flexibility is achieved, since each environment can define its own local policies to rule interactions.

3.1. The Role level. In our proposal, the upper level of an infrastructure is composed by a set of roles related to the same application context. Such definition implies two important features of an infrastructure:

- it is not bound to specific agents, which can belong to whatever applications, can have their own tasks and can be designed and implemented separately from the environment;
- it can host agents, providing a "wrapper" that not only accepts them, but also assigns them capabilities and a specific behaviour.

In Figure 3.1 the set of roles is represented by a "table" where each role is represented by a "hole", in which an agent can place itself in order to play such role. There could be several "holes" of the same role, if an environment can host more than one agent playing the same role.

This idea of infrastructure enforces the locality concept previously introduced. In fact, each environment can decide how to organize the local hosting of agents and, by defining also mechanisms and policies, how local interactions are ruled.

3.2. The Policy & Mechanism Level. This level deploys the policies that rule the local environment, and provides the mechanisms for the interactions both between agents and between agents and the resources of the environment. While the previous level can be considered as the interface of an environment toward the external world, this level enacts the environment's laws. The simplest example of policy is to allow or deny an interaction between two given roles. Thanks to their autonomy and reactivity, agents can handle situations where something is forbidden by local rules without giving up and aborting their job.

Even if different from policies, we include mechanisms at this level because, as policies, they enable the interactions between roles and with local resources. To be more precise, this level should be split into two sub-levels: a policy one and a mechanism one, since policies rely on mechanism; but from our point of view this distinction is not relevant.

3.3. The Resource Level. At the lowest level we can find the resources local to an environment. They can be legacy resources that are hard to change or affect. So, it is important that the policy & mechanism level makes them available in a useful format for agents. As mentioned, this level is considered as part of the infrastructure, but modelling resources as outside the infrastructures does not mine the proposed model.

Also in this case, the use of roles helps in abstracting from the single agent or application, because mechanisms has to be enacted for a generic role, covering the wide range of actual agents that play such role.

Our proposal permits to disregard how local resources are managed, providing that appropriate access mechanisms are supplied.

4. Implementing Role-based Infrastructures. The infrastructure abstraction must have its concrete counterpart, i. e., an appropriate implementation. We are currently exploring the implementation of the proposed role-based infrastructures exploiting RoleX [3], which is a role-based interaction system implemented in the context of the BRAIN framework [10]. RoleX considers roles as first-class entities and enables agents to dynamically assume, use and release roles; the idea behind its implementation comes from the Aspect Oriented Programming [11], even if it has been re-adapted to the agent scenario. Moreover, RoleX is inspired by real life, thus provides a high degree of freedom to agents that want to assume and play roles. The real life inspiration of RoleX is emphasized by the role *external visibility* that it provides: the role an agent is playing can be identified directly from an external point of view (e.g., another agent) without requiring to ask the interested agent for it. RoleX is a pure Java middleware and is not another agent platform. RoleX can be easily associated with any Java agent platform. RoleX adopts the BRAIN XML notation to describe roles. The adoption of XML provides several advantages, as described later.

We have chosen RoleX for three main reasons.

First, being part of the BRAIN framework, it supports the different phases of the software development, from design to implementation [4], and we think this is a valid help to developers, which can have coherent development phases and not fragmented solutions. Moreover, RoleX supports also dynamic assumption of roles at runtime, granting a high degree of flexibility.

Second, we aim at exploiting the XML notation to produce definitions of roles that are interoperable, flexible, and expressive. Each role can be defined by means of an XML document compliant to a given XML Schema, which is independent of the language of its implementation; then XML allows developers to express information in a tagged and structured way, both human- and machine-readable; finally, XML documents can be translated into other formats (e. g., HTML) in order to be catalogued or viewed in a more suitable way.

Third, RoleX provides some mechanisms to define interaction rules, and this turns out useful to define the policies of the infrastructures. Among such mechanisms, we can mention some flexible security mechanisms [5], which are based on the Java Authentication and Authorization System (JAAS) architecture [28], and enable a fine-grain control over the allowed operations of roles. Then, a simple yet useful mechanism permits to define which roles can interact with which other roles; this is useful, for instance, to avoid collusion in an auction context. Finally, perhaps trivial, an agent can assume a role only if it has the right permissions.

RoleX has been exploited also to implement *computational institutions* [6], where it has proved to be a powerful and flexible means to implement abstractions.

4.1. Role Description and Implementation. To grant a high level of abstraction in deciding which role to play, and to cope with dynamic situations, it is important to uncouple roles from their implementation; this allows agents to focus on the semantics of the roles, rather than their code. To this purpose RoleX uses *role descriptors*, which are entities that describe a role, for example by means of information such as keywords, a contest, an aim, a version, a creation date and any further needed piece of information. Descriptors are defined by XML documents written exploiting notation of BRAIN. Besides relieving programmers of the knowledge of role implementation, the descriptors are useful also to hide to the agent the physical location of the role implementation, to enable role composition, to change role implementation in a transparent way, and to allow the *agent* programmers to disregard about the work of *role* programmers and viceversa.

Since the agents are developed using Java, our implementation enables automatically the translation from the XML documents representing descriptors to a set of Java classes. In this way, an agent can directly access the descriptors without needing an XML-parser. The Java implementation of a role is composed of two parts: a Java interface (*role interface*) and a Java class (*role implementation*); the interface provides external visibility (simulating multiple inheritance) while the class provides the effective role behavior.

A role assumption means that the RoleX system performs run-time bytecode manipulation, and in particular (i) adds each role class member (both methods and fields) to the agent class, in order to add the set of capabilities of the role and, at the same time, (ii) forces the agent class to implement the role interface, in order to modify its appearance and to allow other agents to recognize it as playing that role.

Since the above mechanism must result in the definition of a new class, our approach exploits a special class loader, called `RoleLoader`, that can change the agent behavior and the external appearance. It is a subclass of SecureClassLoader, which allows us to work in compliance with the Java Security Manager. After the `RoleLoader` has successfully carried out the role assumption process (i. e., the addition of the members and the interface), it can reload the agent restarting it. The assumption process can be briefly described by Figure 4.1, where the agent searches a role repository for an appropriate role descriptor, for example by using keywords, and then asks the `RoleLoader` to reload itself with the chosen role among the retrieved ones. The `RoleLoader` retrieves the implementation corresponding to the role descriptor and adds it to the agent. If everything goes right, the `RoleLoader` sends the new agent an event (called reload event) to indicate that the agent has been reloaded. After the reload event, the agent can resume its execution. Releasing a role is similar to the above process, but this time the `RoleLoader` removes each role member and the role interface, reloading the agent without them.

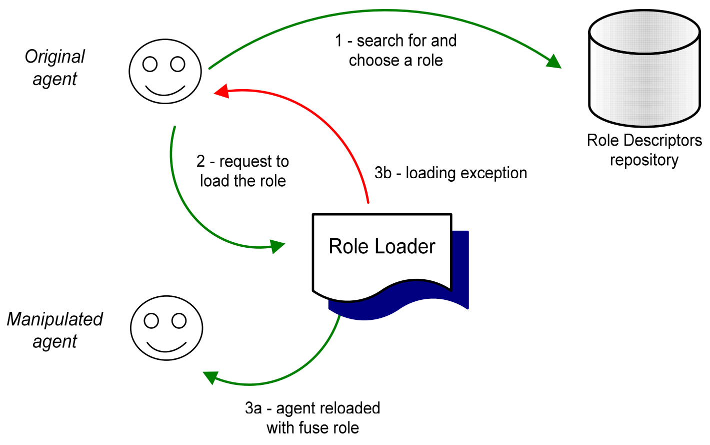

Original agent
1 - search for and choose a role
Role Descriptors repository
2 - request to load the role
3b - loading exception
Role Loader
Manipulated agent
3a - agent reloaded with fuse role

Fig. 4.1. *Role assumption process*

The exploitation of RoleX in the definition of role-based infrastructures will be explained in the next section, by means of a concrete example.

5. Application Examples. This section presents a couple of examples of applications where a role-based infrastructure is defined with the corresponding policies and mechanisms.

5.1. Auctions. The first example relates to auctions. Auctions represent an interesting negotiation means in the agent area, and we have already exploited them in this paper to support some concepts. In an auction there are entities (called *sellers*) that make goods/resources available and entities (called *bidders*) that are interested in using/acquiring such goods/resources. Moreover, there are intermediate entities (called *auctioneers*) in charge of actually performing the negotiation. The price of the resources sold by sellers via an auction is not fixed, but it is dynamically determined by the interest of the bidders [1].

We can figure out that agents negotiate resources or goods via auctions, at given sites representing auction houses [25]. Of course, the way the sellers, the bidders and the auctioneers interact is not bound to a given application or to a given environment, and so they can be considered roles that whatever agent can assume. In this case, the choice of the roles is tightly driven by the kind of application: the *bidder*, the *seller* and the *auctioneer* (see Figure 5.1). So the former step of Section 3 is accomplished.

Figure 5.2 and 5.3 report possible XML descriptions of the bidder and the seller role respectively. The site is in charge of providing role implementations, which are likely to depend on the local policies or mechanisms. From the agent point of view, these descriptions can be exploited to search for an appropriate role in a site and, once found, it can assume the role, placing itself in the “hole” of the site. From the site point of view, the roles made available represent the interface by which the environment can host agents; moreover, roles can be implemented and managed meeting local requirements and in compliance with local polices. In fact, the latter step relates to the choice of the local policies of interaction between roles and between roles and the environment resources.

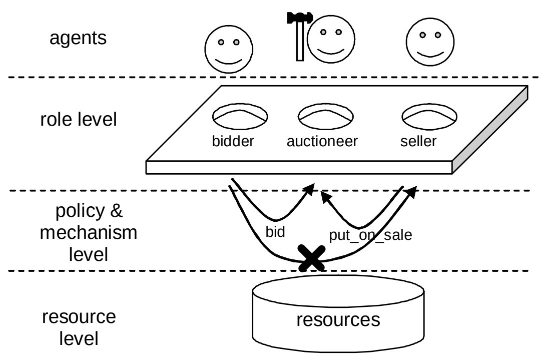


Fig. 5.1. *A role-based infrastructure for an auction house*

```
<?xml version="1.0" encoding="UTF-8"?>
<role xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="XRole.xsd">
 <name>bidder</name>
 <context>auction</context>
 <description>
 This role is the bidder of an auction.
 </description>
 <keyword>auction</keyword>
 <keyword>bidder</keyword>
 ...
 <action>
 <name>bid</name>
 <description>Makes a bid.</description>
 <content>
 <description>Bid.</description>
 <type>Price</type>
 </content>
 </action>
 ...
 </role>
```

Fig. 5.2. *A bidder role defined in XML*

There can be several reasons to have different policies or mechanisms. For example, in one environment the bidders could be allowed to talk each other, while in another environment they cannot, to avoid collusions; this permits to impose local rules or social conventions [32]. For instance, Figure 5.4 reports the possible allowed interactions expressed by a grid whose definition is enabled by RoleX. As in Figure 5.1, bidders and sellers are not allowed to interact directly.

Another reason could be the different implementation of the auction mechanisms: Figure 5.1 shows a message-passing oriented implementation, where, for instance, the bidder agent can bid by sending a message to the auctioneer agent. But if the implementation of the bidding mechanism is based on another model, the local policies must be different. For instance, if the auction relies on a data-oriented model such as tuple spaces [7], the bidding action is implemented as writing information in the local interaction space, as shown in Figure 5.5. This example shows that the same set of roles can be adapted to different implementations.

5.2. Restaurants. This example is taken from the human life, and may not be related to a real application based on agents; however, it is meaningful to understand how roles can be defined and how the interactions among them can be established.

In this example, a node represents a single restaurant. We define three roles: the *customer*, the *waiter*,

```
<?xml version="1.0" encoding="UTF-8"?>
<role xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="XRole.xsd">
 <name>seller</name>
 <context>auction</context>
 <description>
 This role is the seller of an auction.
 </description>
 <keyword>auction</keyword>
 <keyword>seller</keyword>
 ...
 <action>
 <name>put_on_sale</name>
 <description>Put a good on sale.</description>
 <content>
 <description>Good.</description>
 <type>String</type>
 </content>
 </action>
 ...
 </role>
```

Fig. 5.3. *A seller role defined in XML*

	Bidder	Seller	Auctioneer
Bidder	No	No	Yes
Seller	No	No	Yes
Auctioneer	Yes	Yes	Yes

Fig. 5.4. *Allowed interactions*

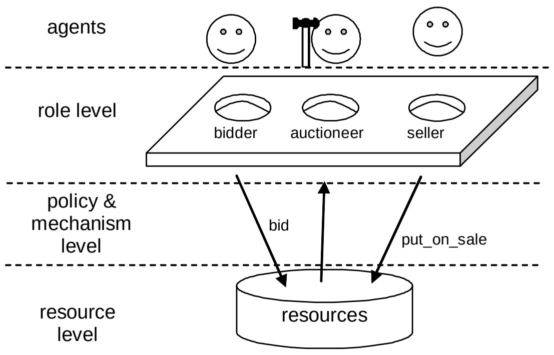


Fig. 5.5. *The same infrastructure relying on a data-oriented model*

and the *chef* (see Figure 5.6). These roles can be thought as instances of the more general roles defined in a client-server model with an intermediate entity (the waiter) between the client (the customer) and the server (the chef), such as several 3-tier solutions.

The role of the customer can have the following capabilities: *ask for the menu*, *order the meal*, *accept the meal*, and *pay the bill.* Note that "*eat the meal*" is not a capability of the role of customer, while it should be of the agent. The waiter role has different capabilities: *take order*, *order the meal* (to the chef), *accept the meal* (from the chef), *give the meal* (to the customer), and *accept the payment.*

Finally, the chef can: *accept an order* and *give the meal.* Again, the cooking of the meal is not an external capability of the chef *role*, but an intrinsic capacity of the chef *agent.*

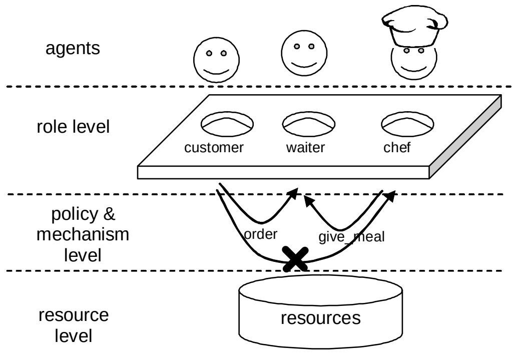


Fig. 5.6. *A role-based infrastructure for a restaurant*

The policy & mechanism level ensure that such interactions occur, for example it grants that the customer orders the meal to the waiter and the chef gives the cooked meal to the waiter. Some interactions may be disabled, such as the direct interaction between the customer and the chef (see Figure 5.6).

Now, let us suppose that the scenario changes. To save money, little restaurants do not have the waiter, but the chef itself is in charge of accepting and satisfying the customers' requests (see Figure 5.7).

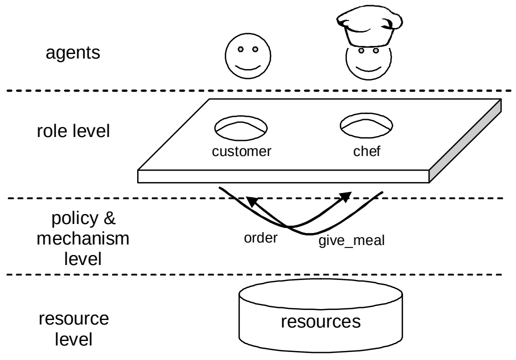


Fig. 5.7. *A restaurant without waiters*

In this case, the infrastructure can exploit the same roles - disregarding of course the waiter role. On the one hand, the agents can assume the same roles as in the previous scenario, and they can perform the same actions in the restaurant. On the other hand, the policy and mechanism level must be changed in order to allow the new interactions, letting the customer give the meal order to the chef, and the chef give the meal to the customer.

This example shows that the same roles can adopt different interaction patterns without affecting the role definitions and, as a consequence, the agents' way to achieve their goals.

The implementation of this example is similar to the previous one.

6. Related Work. In this section we report some role-based approaches for agents.

Elizabeth Kendall has done a lot of work about roles and agents [21], and her proposal is perhaps the most complete. Her effort aims at covering different phases of the agent-based application development, from the analysis to the implementation. To achieve this, her proposal exploits both Object Oriented and Aspect Oriented Programming (AOP) [11] in order to make flexible, reusable and dynamic the use of roles. With regard to role-based infrastructures, an interesting proposal is represented by *role model catalogues*, which are collections of role models used in agent applications. Even if not thought for infrastructures, they can be exploited to group roles belonging to the same application context.

AALAADIN [15] is a role approach that exploits a meta-model to define models of organizations. Such meta-model is based on three main concepts: *agent*, *group* and *role*. The concept of group can be exploited to model the infrastructures proposed in this paper. A drawback of this approach is that it covers only the analysis and design phase, while no support for the implementation phase is provided, which can turn useful to have coherence in the entire development process.

Yu and Schmid [31] exploit roles assigned to agents to manage workflow processes. They traditionally model a role as a collection of rights (activities an agent is permitted on a set of resources) and duties (activities an agent must perform). An interesting issue of this approach is that it aims to cover different phases of the application development, proposing a *role-based analysis* phase, an *agent-oriented design* phase, and an *agent-oriented implementation* phase, that is partially supported. Joining this approach with some means to define infrastructures can lead to a complete and interesting proposal.

The Tractable Role-based Agent prototype for concurrent Navigation Systems (TRANS) is a multi-agent system with support for role and group behaviours [16]; this approach explicitly takes into consideration that agents can be mobile. Moreover, TRANS allows the definition of rules on the role assumption by agents, such as priority, exclusivity, compatibility and the distinction between permanent and temporary roles. Even if designed for a specific application scenario, TRANS provides an interesting role model and, in particular, a support for group of roles. A possible drawback is that the implementation lacks of dynamism, and thus it is not appropriate for agent applications for dynamic and unpredictable scenarios.

Another interesting role approach is GAIA [30]. The GAIA main aim is modelling agent systems as *organizations* of agents, where different roles interact. In GAIA, roles are considered only in the analysis phase, and a role definition includes concepts like responsibilities, permissions, activities and interaction protocols. GAIA focuses on role interactions and supports a model to describe dependencies and relationships between different roles. The above model is supported at the design phase and is made of a set of *protocol definitions*, each of them related to the kind of interaction among roles. Similarly to the above approaches, even GAIA lacks on the implementation support and provides a quite strict definition of role related concepts, leaving freedom to developers.

Maria Fasli made a proposal that joins several concepts, such as commitments and obligations, with the powerful of roles, promoting the use of a formal notation and analysis of the applications [14]. The base idea of this proposal is that multi-agent systems are composed of *social agents*, which are social since they do not act isolated. The term *social agent* refers either to a single agent or to a group of correlated agents. In fact, social agents' decisions and acts can affect those of other agents, even if not intentionally, and thus this proposal presents a formal definition of the agent structure using roles and relationships among them. The assumed role defines the "social position" of the joining agent in the social agent, so that each agent in the social agent knows its position and plays accordingly to it. Even if this proposal offers a comprehensive view of social and collective activities, describing them as social and collective concepts, there is no concrete support at the implementation level.

The Role/Interaction/Communicative Action (RICA) theory [26] was born with the main aim of improving the FIPA standard with support for social concepts. The RICA theory fuses communication concepts with social ones, and in particular merges FIPA-ACL and organizational models, keeping all concepts as first class entities. The RICA theory recognizes *communicative entities*, which are those that can be aggregated and organized in a social way; in other words communicative entities are agents acting in a society. Each agent (type) is defined through the role (types) it will play; each role can perform one or more action, implementing a social behaviour. Action can be specialized in social and communicative ones, which emphasizes the above statement. Finally, each role can be specialized as a social role, which represents the behaviour of agents interacting in a social context. The RICA metamodel can be used as a formal language, providing support for the analysis and design phase of an agent application. Thanks to the RICA-J (RICA Jade) implementation, this proposal is complete and can be used during all phases of application development. Probably the most important drawback of this proposal is that it re-defines the concept of agent, leading to a possible incoherence with other agent theories and approaches.

In [33] Haibin Zhu describes a role model that is tied to both the computer and human parts involved in collaborations, and in particular tries to provide help to human in computer-supported collaborations. This model defines a role as a set of *responsibilities* and *capabilities* a human holds. The key of this proposal is the concept of *role agent*, which is an agent with a role attached. An agent that wants to participate in a collaboration must own at least one role. Role agents should help the human during the collaboration, for example helping her to send messages to other agents or entities in the collaborative system (e.g., all agents in the same group). Role agents can change their role at run-time, leading to a dynamic environment and promoting the use of specific roles for specific tasks. Each human must log in the system with a particular role, so she is bound to a role agent, which produces objects as result of the collaboration and/or human wills. This proposal exploits a formal notation, which emphasizes that a role is defined not as an object but as a set of messages (both outgoing and incoming) that the role agent can send/receive during the collaboration. Most of the concepts of this proposal, such as groups and messages, are not new, and the definition of roles as set of messages is not really flexible. Nevertheless this proposal well reflect a real situation, and being based on a formal notation, can help designers and developers planning the system.

7. Conclusions and Future Work. Multiagent systems must be carefully designed and developed. This paper has proposed an approach to agent composition based on roles, which are exploited to build infrastructures for agents. This permits to face the development at different levels, ruling both capabilities and restrictions, and enabling the implementation of policies and mechanisms depending on the local context.

The adoption of the RoleX system has had a twofold advantage. On the one hand, it has enabled the concrete implementation of the approach; the proposed example has shown how the different levels of the role-based infrastructures can be implemented or supported. On the other hand, RoleX has exhibited a great degree of flexibility, allowing the uncoupling between role descriptions and role implementations.

With regard to future work, we are exploiting the RoleX system to implement role-based infrastructures for agents, as already mentioned. RoleX enables the dynamic assumption of roles by agents, letting environments define role repositories with their own implementation of roles. We are going to study on the one hand the applicability of the proposed role-based infrastructures, and on the other hand the flexibility of the RoleX system.

Then, effective tools in the field of software engineering are to be developed to support the building of infrastructures. They can help both the environment developers and also the environment administrators, which can decide to change the local policies or mechanisms.

Acknowledgments. This work was supported by the European Community within the project CAS-CADAS.

The author thanks the anonymous authors for their precious comments.

REFERENCES

[1] Agorics, Inc., *Going, going, gone! A survey of auction types*, `http://www.agorics.com/new.html` 1996.
[2] D. Baumer, D. Ritchie, W. Siberski, and M. Wulf, *The Role Object Pattern*, Proceedings of the 4th Pattern Languages of Programming conference (PLoP), Monticello, Illinois, USA, September 1997.
[3] G. Cabri, L. Ferrari, and L. Leonardi, *The RoleX Environment for Multi-Agent Cooperation*, The 8th International Workshop on Cooperative Information Agents (CIA), Erfurt, Germany, Lecture Notes in Artificial Intelligence No. 3191, Springer-Verlag, September 2004.
[4] ———, *Supporting the Development of Multi-Agent Interactions via Roles*, the 6th International Workshop on Agent-Oriented Software Engineering (AOSE) at AAMAS 2005, Utrecht, The Netherlands, July 2005.
[5] ———, *Applying Security Policies Through Agent Roles: a JAAS Based Approach*, Science of Computer Programming, (Elsevier, Amsterdam-NL), Vol. 59, No. 1-2, January 2006, pp. 127–146.
[6] G. Cabri, L. Ferrari, and R. Rubino, *Building Computational Institutions for Agents with RoleX*, technical report MO/AG/05/001.
[7] G. Cabri, L. Leonardi, and F. Zambonelli, *Auction-based Agent Negotiation via Programmable Tuple Spaces*, Proceedings of the 4th International Workshop on Cooperative Information Agents (CIA 2000), LNCS No. 1860, Boston (USA), July 2000.
[8] ———, *Engineering Mobile Agent Applications via Context-dependent Coordination*, IEEE Transactions on Software Engineering, Vol. 28, No. 11, November 2002, pp. 1040–1056.
[9] ———, *Modeling Role-based Interactions for Agents*, The Workshop on Agent-oriented methodologies at the 17th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2002), Seattle, Washington, USA, November 2002.
[10] ———, *BRAIN: a Framework for Flexible Role-based Interactions in Multiagent Systems*, The 2003 Conference on Cooperative Information Systems (CoopIS), Catania, Italy, November 2003.
[11] *Communication of the ACM, Special Issue on Aspect Oriented Programming*, Vol. 33, No. 10, October 2001.
[12] S. Clearwater, *Market-based Control: a Paradigm for Distributed Resource Allocation*, World Scientific, 1995.
[13] B. Demsky and M. Rinard, *Role-Based Exploration of Object-Oriented Programs*, Proceedings of the International Conference on Software Engineering 2002, Orlando, Florida, USA, May 19-25, 2002.
[14] M. Fasli, *Social Interactions in Multi-Agent Systems: A Formal Approach*, The First European Workshop on Multi-Agent Systems (EUMAS), 18-19 December 2003, Oxford, UK

[15] J. Ferber and O. Gutknecht, *AALAADIN: A meta-model for the analysis and design of organizations in multi-agent systems*, in Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS'98), 1998.

[16] S. Fournier, D.Brocarei, T.Devogele, and C. Claramunt, *TRANS : A Tractable Role-based Agent Prototype for Concurrent Navigation Systems* , The First European Workshop on Multi-Agent Systems (EUMAS), 18-19 December 2003, Oxford, UK.

[17] M. Fowler, *Dealing with Roles*, `http://martinfowler.com/apsupp/roles.pdf` 1997.

[18] N. R. Jennings, *An agent-based approach for building complex software systems*, Comm. of the ACM, Vol. 44, No. 4, 2001, pp. 35–41.

[19] N. R. Jennings, M. Wooldridge, eds., *Agent Technology: Foundations, Applications, and Markets*, Springer-Verlag, March 1998.

[20] N. M. Karnik and A. R. Tripathi, *Design Issues in Mobile-Agent Programming Systems*, IEEE Concurrency, Vol. 6, No. 3, July-September 1998, pp. 52–61.

[21] E. A. Kendall, *Role Modelling for Agent Systems Analysis, Design and Implementation*, IEEE Concurrency, Vol. 8, No. 2, April-June 2000, pp. 34-41.

[22] B. B. Kristensen and K, Øterbye, *Roles: Conceptual Abstraction Theory & Practical Language Issues*, Special Issue of Theory and Practice of Object Systems on Subjectivity in Object-Oriented Systems, Vol. 2, No. 3, 1996, pp. 143–160.

[23] *Compact Oxford English Dictionary online*, `http://www.askoxford.com`

[24] R. S. Sandhu, E. J. Coyne, H. L. FeinStein, and C. E. Youman, *Role-based Access Control Models*, IEEE Computer, Vol. 20, No. 2, 1996, pp. 38-47.

[25] T. Sandholm and Q. Huai, *Nomad: Mobile Agent System for an Internet-Based Auction House*, IEEE Internet Computing, Special issue on Agent Technology and the Internet, Vol. 4, No. 2, March-April 2000, pp. 80–86.

[26] J. Manuel Serrano and S. Ossowski, *On the Impact of Agent Communicative Languages on the Implementation of Agent Systems*, Cooperative Information Agents VIII, M.Klush, S. Ossowski, V. Kashyap, R. Unland eds., Lecture Notes in Artificial Intelligence, ISSN 0302-9743, Springer, 2004

[27] B. Shneiderman and C. Plaisant, *The Future of Graphic User Interfaces: Personal Role Manager*, Proceedings of the conference on People and computers IX, Glasgow 1994

[28] SUN Microsystems, *Java Authentication and Authorization Service (JAAS)*, `http://java.sun.com/products/jaas/`

[29] A. Tripathi, T. Ahmed, R. Kumar, and S. Jaman, *Design of a Policy-Driven Middleware For Secure Distributed Collaboration*, Proceedings of the 22nd International Conference on Distributed Computing System (ICDCS), Vienna (A), July 2002.

[30] M. Wooldridge, N. R. Jennings, and D. Kinny, *The Gaia Methodology for Agent-Oriented Analysis and Design*, Journal of Autonomous Agents and Multi-Agent Systems, Vol. 3, No. 3, 2000, pp. 285–312.

[31] L. Yu, and B. F. Schmid, *A conceptual framework for agent-oriented and role-based workflow modelling*, in Proceedings of the 1st International Workshop on Agent-Oriented Information Systems, G. Wagner and E. Yu eds., Heidelberg, June 1999.

[32] F. Zambonelli, N. R. Jennings, and M. Wooldridge, *Organizational Rules as an Abstraction for the Analysis and Design of Multi-agent Systems*, Journal of Software Engineering and Knowledge Engineering, 2001.

[33] H. Zhu, *A Role Agent Model for Collaborative Systems*, Proceedings of the 2003 International Conference on Information and Knowledge Engineering (IKE'03), Las Vegas, Nevada, USA, June, 2003, pp. 438–444.

Edited by: Shahram Rahimi, Raheel Ahmad
Received: December 9, 2005
Accepted: March 19, 2006

Scalable Computing: Practice and Experience
Volume 7, Number 1, pp. 49–58. http://www.scpe.org
ISSN 1895-1767
© 2006 SWPS

SOFTWARE AGENTS IN SUPPORT OF STUDENT MOBILITY

MARIA GANZHA*, WOJCIECH KURANOWSKI†, AND MARCIN PAPRZYCKI‡

Abstract. Autonomous software agents are often claimed to become a new generation of tools facilitating efficient management of information. While a number of possible agent application areas can be found in the literature, support for "academic mobility" is not one of them. At the same time student mobility is one of the important objectives within the European Union and, as we argue in this paper, software agents could be used to streamline administrative processes involved in setting up student participation and help students that are interested in it as well as administrative units that have to support it. In this paper we introduce an agent system designed to facilitate student mobility, present UML diagrams of agents of that system and discuss an initial implementation of a system-skeleton.

Key words. Multi-Agent System, Agent mobility, JADE agent environment

1. Introduction. One of the more important current goals that the European Union is striving at achieving (with only limited success) is social mobility. In this context, one of promising ways of achieving future social mobility is through various forms of "academic mobility" involving students and faculty members of EU-located institutions of higher learning visiting other such institutions. Mobility of "academicians" is supported financially through a number of Marie Curie Mobility Programs. There, programs like Socrates and Mundus are designed, among others, to allow students to visit universities in other EU countries and spend there one or two semesters, while obtaining a living stipend from the EU. Such a visit is possible when: (a) two universities have a bilateral agreement and (b) student applies to the program and is accepted (if there are more interested students than the agreed number of exchanges, wins a competition). Note that faculty members can be also a part of Socrates/Mundus agreements and therefore results presented here can be extended into support of faculty mobility, but they are outside of scope of our current interest.

Obviously, arranging a student visit involves a large number of administrative steps and further steps are also required post completion of a visit. Fulfillment of all necessary requirements is a tedious task and takes a lot of energy on the part of the student and resources on the part of the University.

As it was suggested in [8, 12, 16] autonomous software agents are one of best possible approaches to manage and deliver personalized information in large complex environment. Recently a few research projects have attempted at pursuing this suggestion. One of important projects in this area is the EU-funded—Pellucid [10, 18, 13, 14, 15]. Pellucid attempted at tackling management of experience in public organizations, particularly those aspects of experience management related to organizational mobility (e.g. movement or circulation of staff from one unit to another—within an organization). The basic metaphor for experience management is that of an intelligent assistant that looks over oneâĂŹs shoulder and answers questions one might have at a particular point of work. Such an assistant detects that an employee is working in a particular context, offers knowledge resources that facilitate her work. To this end, the Pellucid platform integrates technologies such as autonomous cooperating agents, organizational memory, workflow and process modeling, and metadata for accessing document repositories [11]. In the context of our work, ubiquitous intelligent access to document repositories and document flow modeling are of particular interest. Results obtained within the Pellucid project were somewhat similar to research on utilizing software agents in document flow reported in [1, 2]. Finally, in [9] a schema of an architecture of the X-DoC WFMS project, which involves conceptualization and implementation of a workflow management system in Graduate Admission Process, was presented.

Following these suggestions we have decided to develop an agent system that would facilitate and support a different aspect of "student management"— SOCRATES-type mobility program(s). Results presented here are an extension of work reported in [3, 4].

We proceed as follows. In the next section we summarize steps that have to be undertaken by a student who would like to participate in a mobility program. We follow with the description of the design of an agent system and details of its implementation and, in Section 4, discuss the performance of the system. We complete paper with a brief description of our future research directions.

*Department of Administration, Elbląg University of Humanities and Economy, ul. Lotnicza 2, 82-300 Elbląg, Poland (ganzha@euh-e.edu.pl),
†Wirtualna Polska, Software Development Department, ul. Traugutta 115C, 80-226 Gdańsk (wkuranowski@wp-sa.pl),
‡Computer Science Institute, SWPS, ul. Chodakowska 19/31, 03-815 Warszawa, Poland (marcin.paprzycki@swps.edu.pl).

2. Student mobility—what has to happen? Let us consider two EU-based institutions of higher learning that are to be involved in a Socrates-type student exchange program. While there exists a number of possible names for such institutions (e.g. college, academia, university etc.), hereafter we will use a name *university* to simplify the description. The first thing that is happen is that two, or more, universities have to sign a bilateral agreement and report it to the "central-Socrates-agency" (in Brussels). It is only after this agreement is officially registered with such an EU-agency when students can be accepted into the program. Since the agreements are typically signed by International Offices of each university, they are in some ways outside of bound of our system (for more details see below).

Administrative steps that lead to student participation in the program involve a number of administrative units within both universities. Nowadays, even in countries like Poland or Romania, we can observe fast increasing role of electronically stored and processed data within universities (e.g. student records). Furthermore some universities already provide an interface that allows students to check items like: course-schedule, upcoming exams, earned credits etc. Finally, almost all students and most faculty members and administrators communicate using e-mail (to a lesser or greater extent). Thus there exist basis for developing system like the one outlined here. Let us now conceptualize situation when a student from an EU-located university wishes to participate in a Socrates-type student exchange program. We assume here that her home university has already a number of bilateral Socrates-agreements signed and registered with the central agency. In this case the following steps have to be completed (see also Figure 2.1):

- before departure
  1. Selecting foreign university
  2. Applying to the program
  3. Being accepted to a particular exchange
  4. Delivering all necessary data appropriate administrative units both at the local and the foreign university
  5. Organizing a place to live at the foreign site
- after arrival at foreign university:
  6. Contacting appropriate department at the host university
  7. Arranging the schedule of courses to be taken
  8. Managing courses and credits required to meet the requirements of the exchange program
- after returning to the home-university:
  9. Completing a survey or delivering a report to the home-site coordinator.

In current practice, the first four steps involve mostly interactions between the student and the Dean's Office at her local university, as well as an information exchange with the local exchange program coordinator. Let us note here, that the situation when multiple students are interested in a limited number of openings within an exchange program is handled in (3) "being accepted to a particular exchange". There a "competition" takes place and an appropriate number of students are selected. What is particularly interesting from our perspective is answer to the question, what happens to these students who did not qualify to a given exchange. As it becomes clear below, our proposed system allows such students, in a very natural way, becoming involved in subsequent "competitions" (if any available exchanges remain unfilled). Step (5) is often completed "automatically" by an office at the host institution that receives information about incoming exchange students as a part of the document circulation involved in steps (1)–(4). Otherwise, student has to search a flat or to communicate with a separate organization which supervises dormitories/apartment rental. After arriving at the chosen university student has to contact the host department to arrange the course schedule in such a way to fulfill the requirements of the program (e. g. to accumulate a required number of credits, to study subject areas that were covered by the bilateral agreement etc.).

In all universities, appropriately prepared to handle exchange students, steps (1)–(4), (6)–(8) or (9) don't present problems when considered independently (even if they are not supported by electronic means of communication and thus unnecessarily tedious). Problems materialize when all steps have to be completed "together" and thus, when various documents have to circulate between different units within university; between different units in different (foreign) universities and, finally, between these units (both local and foreign) and the student. Moreover, since not every university supports electronic data management to the same extent (and some universities in countries like Bulgaria, have only a very minimal IT support in administration), it is often the case that an extremely large number of documents have to be transferred "manually". This involves sending letters, faxes, receipts (in case of organizing a flat) and/or numerous telephone calls. In it particularly in this

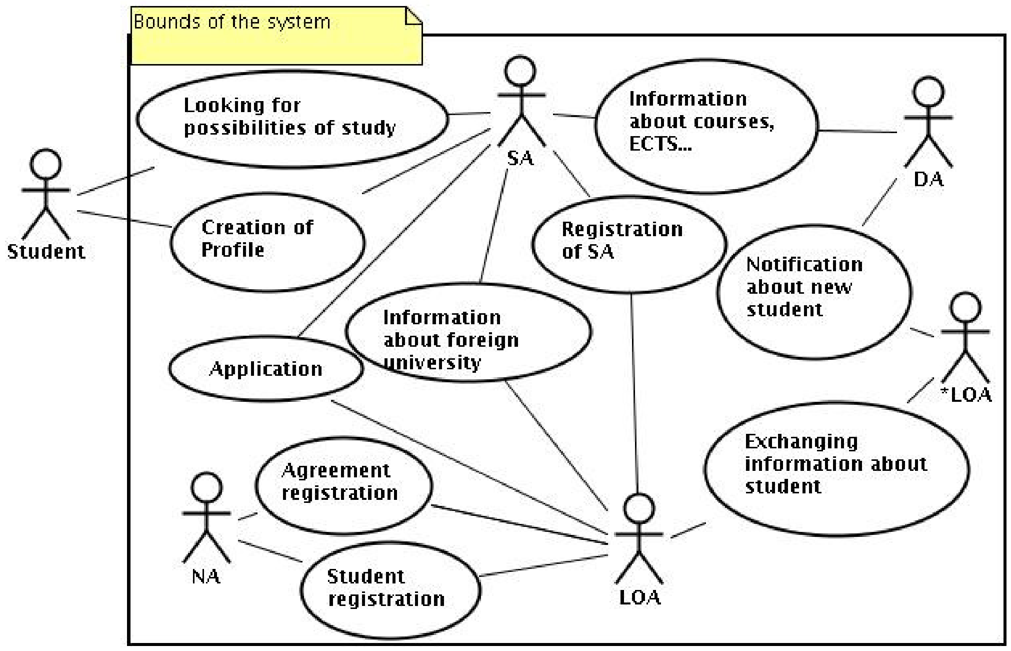

Bounds of the system
Student
SA
DA
*LOA
NA
LOA
Looking for possibilities of study
Creation of Profile
Information about courses, ECTS...
Registration of SA
Notification about new student
Application
Information about foreign university
Exchanging information about student
Agreement registration
Student registration

Fig. 2.1. *System Summary; interactions between agents*

regard that the proposed system, described in the next section, is expected to be particularly helpful.

3. Student mobility — proposed agent system. The main idea of our project is to develop a solution which would make formalities of taking part in a student exchange program simpler, and also reduce number of issues that presently have to be dealt with "face-to-face." We propose a system that would facilitate semi-automatic (and possibly even automatic) decision making and enable fully automatic flow of information required to establish participation in an exchange program. Furthermore, as the system develops, it could completely remove humans from the process (other than the student expressing a desire to participate in it). Let us start from summarizing (in Figure 2.1) the proposed flow of activities. Here, we have divided the functionalities into the following agents:

Student Agent (*SA*) is an interface between the student and the system and is also students' "representative." Following the line of reasoning presented in [3, 4, 10] it is assumed that in the university of the future the *SA* will be able to organize or provide view of students' schedule, check the total number of credits acquired thus far, make an appointment with a professor and/or advisor etc. In this way the *SA* is a limited case of well-known (in agent literature) paradigmatic concept of a "personal agent" [12]. In our current system, the *SA*, plays an even more limited role and represents the student only in organizing his participation in the student-exchange program. After the student is accepted and arrives at the foreign university, the *SA* communicates with the *Department Agent* at the host university and supplies the exchange student with all required information. Among others, it helps student to arrange his course schedule. While in Figure 2.1 we can see the top level view of all interactions with other agents that the *SA* is involved in, in Figure 3.1 we present the complete UML state diagram of this agent. The MCDM stands for Multicriterial Decision Making (the same demarcation is used also in the case of the Local Office Agent) and denotes the fact that in a full-blown, mature system implementation this step of agent operation involves an optimization procedure that leads to a decision. In the case of the *SA* the decision where to study could involve a very large number of criteria such as, geographical location (e. g. student wants to go where climate is warm), particular country (e.g. student does not want to go to France), program of study (e. g. student is interested in e-commerce and not in theoretical foundations of computer science) etc. Note that the blue (grey) box Studying involves a large number of steps (the same notation is used across the paper). Inside of the Studying box, one more MCDM is enclosed. This one involves selection of class schedule. Here, among others, decisions balancing interest in subject with willingness to wake up at 7 AM could be made.

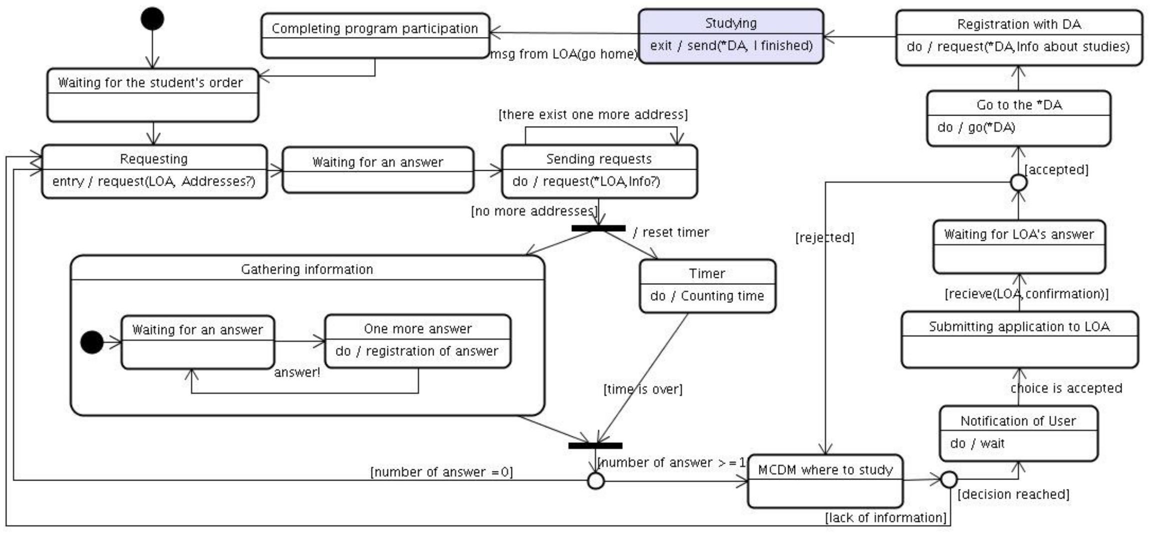


Fig. 3.1. *Student Agent State diagram*

In our system, the *Local Office Agent* (*LOA*) acts as a coordinator of the Socrates program (and even its diagram shows this by indicating that in most part the *LOA* "services" received messages). *LOA* stores information about universities that currently have bilateral agreements with its university. This list is constructed on the basis of messages obtained from the Notification Agency agent. Here we have to recall that signing bilateral agreements is the domain of International Offices. The way our system works, these offices have to notify the Notification Agency agent first and that agent has to notify the *LOA* that it is ready to accept students within the purview of a given exchange (such a notification contains also all necessary information, including appropriate deadlines). Otherwise it would be possible that the *LOA* would accept students to the program that the Notification Agency would not yet be ready to service. The *LOA* exchanges appropriate messages required to set up departure of a student to another university and handles student returning back home form an exchange. In Figure 3.2 we present the complete UML state diagram of this agent. Note the Considering Applications, box (appearing within that Figure). In the proposed system Student applications are accepted until a certain deadline. When the deadline passes, they are pre-processed first to eliminate students who do not satisfy initial selection criteria (e.g. at a given University students who have not completed successfully previous semesters may not be allowed to participate in the exchange program). The remaining applications are considered using an MCDM, the details of which are likely to be institution dependent (e. g. at a given University, the Grade Point Average (GPA) in the core courses may be more important than the overall GPA).

Notification Agency (*NA*) represents offices ("in Brussels") that supervise the student exchange program (including the financial matters). In our system, the *NA* has two functions: (1) the above described bilateral agreement management; each such agreement has to be registered with the *NA* that in turn notifies the *LOA* and the $*LOA$ that it is ready to service it, and (2) student participation management. Specifically, the *NA* has to be notified that a given student is to participate in an exchange program. In response the *NA* validates the proposal to assure that it adheres to the rules of the program (also to check if the limits of participation in a given program have not been somehow breached). When a given proposal has been positively validated (1) one of the spots available in the negotiated bilateral agreement is taken and (2) a given student will be funded by the Socrates scholarship. In Figure 3.3 we present a complete UML diagram of the *NA* agent.

Department Agents (*DA*) may be conceptualized as a virtual combination of a department head and secretary. One such agent is created for each individual departments of each university. These agents are envisioned to be responsible for courses offered during a given semester, course schedules, and calculation of ECTS, etc. Since most of functionality of this agent is related to the functioning of the university rather than to our system and falls mostly beyond the scope of our work, we have decided to omit its detailed UML-based conceptualization.

Finally, the $*$*Local Office Agent* ($*LOA$) is the *LOA* counterpart at the foreign host institution. In other words, the $*LOA$ is the *LOA* of the foreign university.

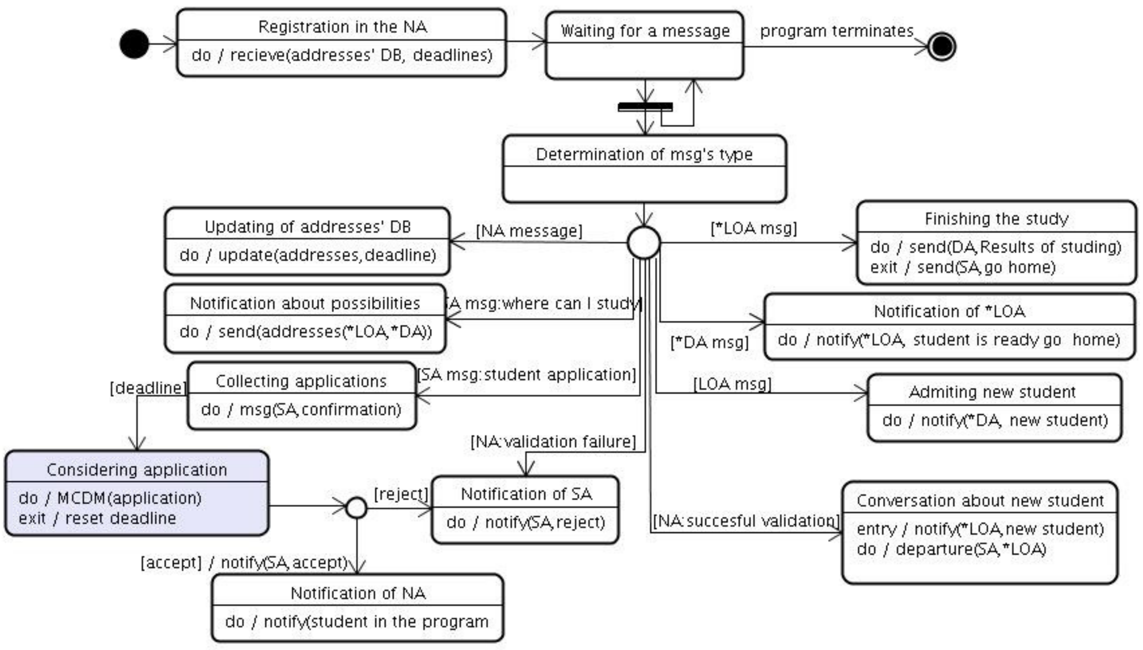

Registration in the NA
do / recieve(addresses' DB, deadlines)
Waiting for a message
program terminates
Determination of msg's type
Updating of addresses' DB
do / update(addresses,deadline)
[NA message]
[*LOA msg]
Finishing the study
do / send(DA,Results of studing)
exit / send(SA,go home)
Notification about possibilities
do / send(addresses(*LOA,*DA))
[SA msg:where can I study]
Notification of *LOA
do / notify(*LOA, student is ready go home)
[*DA msg]
[deadline]
Collecting applications
do / msg(SA,confirmation)
[SA msg:student application]
[LOA msg]
Admiting new student
do / notify(*DA, new student)
[NA:validation failure]
Considering application
do / MCDM(application)
exit / reset deadline
[reject]
Notification of SA
do / notify(SA,reject)
[NA:succesful validation]
Conversation about new student
entry / notify(*LOA,new student)
do / departure(SA,*LOA)
[accept] / notify(SA,accept)
Notification of NA
do / notify(student in the program

Fig. 3.2. *Local Office Agent State diagram*

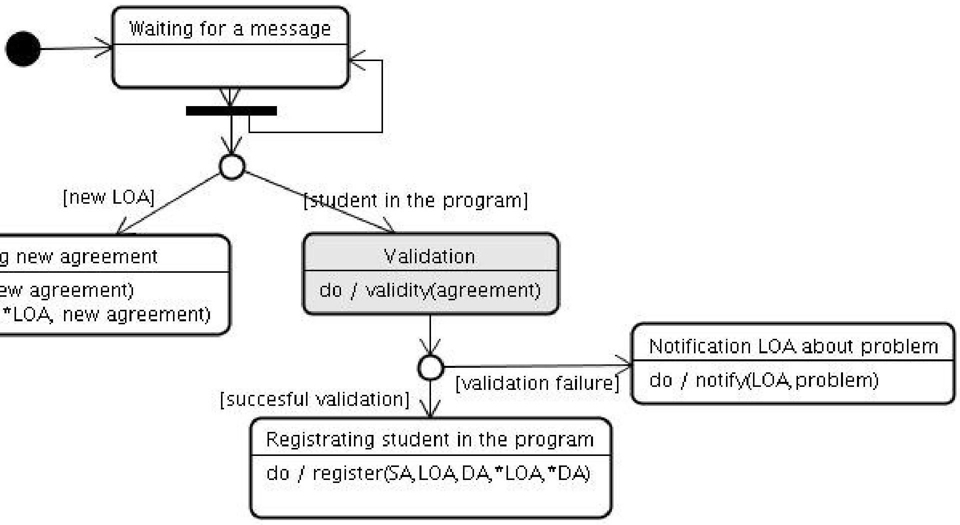

Waiting for a message
[new LOA]
[student in the program]
Registrating new agreement
do / input(info new agreement)
exit / send(LOA,*LOA, new agreement)
Validation
do / validity(agreement)
Notification LOA about problem
do / notify(LOA,problem)
[validation failure]
[succesful validation]
Registrating student in the program
do / register(SA,LOA,DA,*LOA,*DA)

Fig. 3.3. *Notification Agency state diagram*

3.1. Agent Interactions. Let us now list interactions between agents that take place when the *SA* attempts at arranging the exchange program for the student (see Figure 2.1). We assume that the system has been initialized, that the *NA* has send the list of confirmed bilateral agreements to the *LOA*´s residing in the system etc. In other words, the system is ready to service students. In this stage, student has communicated with her *SA* and established the selection criteria (e.g. country, subject area, etc.). Then, the system performs the following actions (working autonomously — as we assume that when student specifies requirements, agents make all decisions). Note that communication between agents is achieved through exchange of standard ACL messages.

1. *SA* sends search request to the *LOA* to get addresses of all foreign universities that her *LOA* has bilateral agreements signed with (in the specified field of study)

2. upon reception of the address list, the *SA* sends messages to all of them, requesting information about local requirements/arrangements/possibilities
3. foreign *DAs* (∗*DAs*) reply providing requested details
4. *SA* performs multicriterial optimization (MCDM) and selects one or more of available universities as the place where the student should go for the exchange
5. *SA* informs the student about possibilities and suggests which one to choose (to be able to run the system automatically we have removed this step and replaced it with a fully automatic selection process)
6. *SA* sends to the *LOA* an application to the selected university and upon receiving confirmation that the application has been received suspends itself until a decision is reached (the *LOA* is assumed to process applications in batches after certain deadlines)
7. *LOA* informs the *SA* if student qualified for the exchange—if student did not qualify, the *SA* goes back to 5-above and the process repeats
8. *LOA* informs the NA that a given student was selected to participate in a given student exchange and waits for confirmation
9. when the *NA* validates the request it confirms it by sending message back to the *LOA*
10. *LOA* sends all of the necessary documents for the student to become a part of the exchange program to the (host) ∗*LOA* and obtains confirmation
11. ∗*LOA* registers an incoming exchange student (her/his *SA* is also registered with the local system)
12. *SA* moves to the foreign host
13. *SA* contacts appropriate ∗*DA*
14. ∗*DA* informs the *SA* about courses available
15. *SA* performs multicriterial optimization and on the basis of knowledge of student preferences and selects courses that match them
16. *SA* informs the ∗*DA* that student completed scheduled courses (currently, to test the system, we have implemented a simple automatic selection, but a realistic system should involve student in the decision-making process; both possibilities are covered by the Studying box in Figure 3.1)
17. ∗*DA* informs the *SA* and the ∗*LOA* how many ECTS student accumulated
18. ∗*LOA* "allows" the *SA* to go home
19. *SA* moves to its home container
20. ∗*LOA* informs *LOA* about results of student exchange program participation (grades, ETCS, etc.)

Obviously, at this stage of the project the multicriteria decision making processes, mentioned above in points (4) and (14), have been replaced with a set of very simplistic selection procedures. However, delving into decision making was not of our current interest and is definitely outside of the scope of this paper. What we were interested was to develop the system skeleton and illustrate experimentally that it works. To show that agents communicate accordingly to the specification and that agent mobility is appropriately utilized to work in unison with proposed student mobility. As illustrated in the next section, we have fully achieved this goal.

4. System implementation and operation. The proposed system has been implemented in JADE 3.3 [7]. In a JADE based agent system, all agents exist within a *platform* that can be spread among multiple computers. Within a platform, agents reside in and move between *containers*. In our experimental setup, every container represents one university. We have inserted *LOA*s and *DA*s into each container (recall that a *LOA* can play a role of a ∗*LOA* depending on the direction of the proposed student exchange). Additionally an *NA* is created in the Main-container (the Main-container is the name used by JADE for the "system" container that is created when JADE platform is started for the first time). After the system is initialized in this way we can create as many *SA*s as we need.

A "single" system run involves an *SA* performing all necessary steps to organize the exchange program for its student-master. As noted, in our current implementation we use very simple selection criteria, i. e. the place where the exchange program was to take place was selected on the basis of only two student preferences: field of study and number of ECTS credits she gathered thus far. An example of a system run is represented in Figures 4.1-4.3 (here the, JADE provided, Sniffer Agent which "records" all messages incoming to and originating from agents, it was told to "sniff," was used to indicate the operation of the system).

In the experiment we observe a sample scenario involving five universities (located at five separate computers): UNIV1, UNIV2, UNIV3, UNIV4, UNIV5. At the UNIV1, *DA*s representing IT and Biology departments have been created. Similarly, at the UNIV2 we see departments of IT and Chemistry, at the UNIV3 depart-

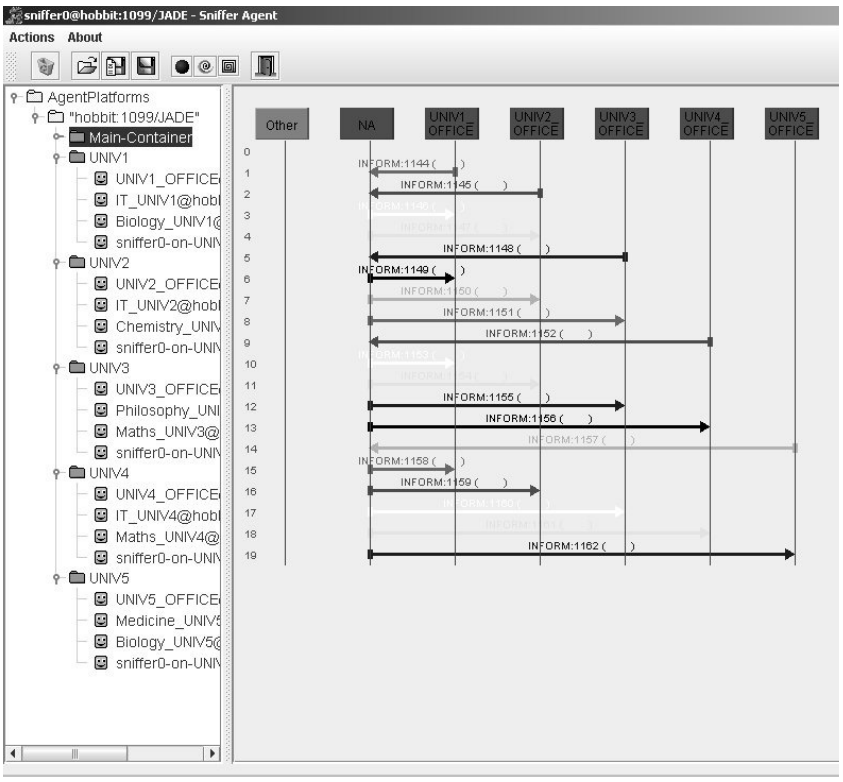


FIG. 4.1. *Sniffer Agent report for the Initial Part of Experiment 1*

ments of Philosophy and Mathematics, at the UNIV4 departments of IT and Mathematics, while at the UNIV5 departments Medicine and Biology. In Figure 4.1 we see the Initial Part of the experiment, where the LOA agents register with the NA. Furthermore, an SA was created within container representing the UNIV1 university. This agent registers with its LOA and later requests addresses of available exchange programs that are of possible interest to its student-master. This process is depicted in Figure 4.2

Finally, in Figure 4.3 we observe the moment when the SA arrives at the UNIV4 university. The main point of this scenario is for an IT student at the UNIV1 to arrange (and complete) an exchange with the IT department at the UNIV4 and this mission is accomplished.

In a separate experiment, using the psexec scripting program [17] we have created 22 containers representing 22 universities (located in 22 countries), on 20 separate, networked computers. We have then placed "random" departments on each one of them and successfully run experiments with "students" (SAs) seeking exchange programs among all of these university departments (computers). A sample screen representing this experiment is presented in Figure 4.4. Finally we have experimented with a "mixed environment." For instance we have run the Main-container on a Linux-based laptop, while the remaining computers have been running Windows. We have observed no problems in any of trial runs. More details of these experiments (involving an earlier, somewhat less sophisticated version of the system) can be found in [3, 4].

5. Concluding remarks. Our project, in its current stage, illustrates the most important (from the point of view of agent system design and implementation) features of system that would enable student mobility automation. Those are: mobility, communication, registration, searching etc. Furthermore, the system skeleton has been implemented and shown experimentally to work (even though, we have to admit, utilizing an extremely simplified sets of rules for decision making, selection etc.). We were able to run experiments on a single network, utilizing up to 20 computers, including mixed Linux-Windows setup and found no problems. One of the important issues that have to be considered when constructing agent systems is that each such a system has to reflect the real world. Our example shows potential of software agents to automate an existing real-world scenario. In the next steps of the development of this system, we will attempt at making it to resemble the reality even more, by focusing on developing and implementing the following features:

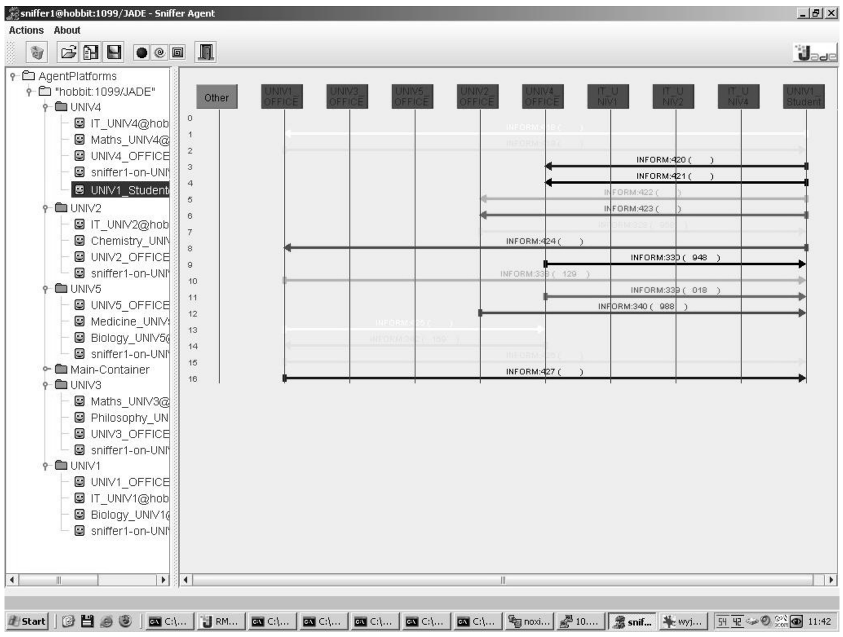


Fig. 4.2. *Sniffer Agent report of* SA *creation (Experiment 1)*

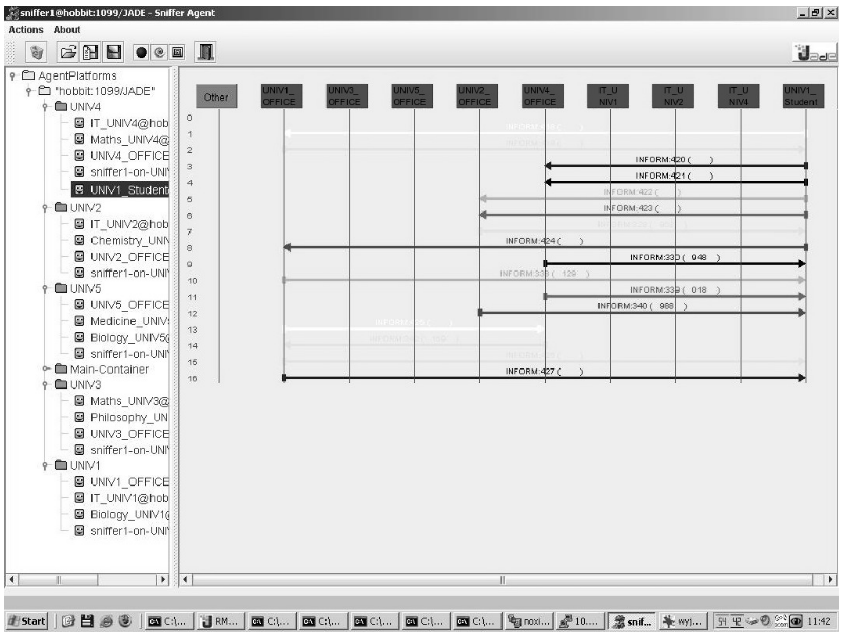
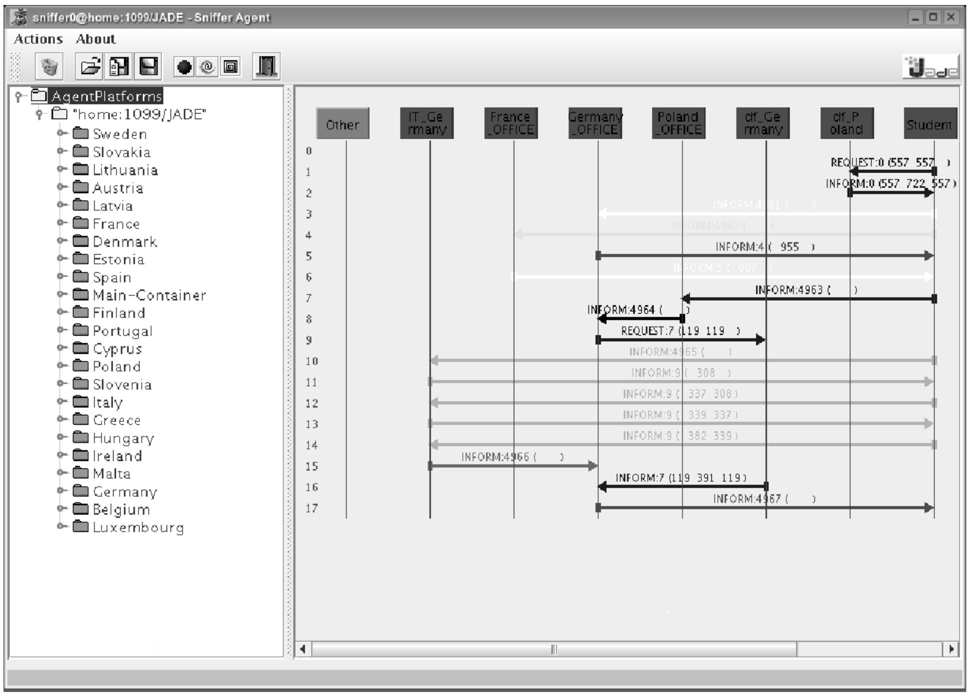


Fig. 4.3. *Sniffer Agent report* SA *arrives at the UNIV4 university (Experiment 1)*

1. Student Agent personalization (agent that actually knows what its student-master really "wants" and is able to truly represent her interests). In this context, we will have to find a way to represent user profile and this representation will have to be tied to the ontologies of "world of academia" that will have to be developed (see 4. below). A proposal how to tie ontologies and user profiles has been recently put forward in [5, 6].
2. Adding functions to the Department Agent that would extend the communication between the *DA* and the *SA* and facilitate possibility of developing the MCDM module that is to select the student-optimal course schedule.



```
0
1  REQUEST:0 (557 557 )
2  INFORM:0 (557 722 557 )
3  INFORM:4961 (    )
4  INFORM:4962 (    )
5  INFORM:4 ( 955 )
6  INFORM:5 ( 007 )
7  INFORM:4963 (    )
8  INFORM:4964 (    )
9  REQUEST:7 (119 119 )
10 INFORM:4965 (    )
11 INFORM:9 ( 308 )
12 INFORM:9 ( 337 308 )
13 INFORM:9 ( 339 337 )
14 INFORM:9 ( 382 339 )
15 INFORM:4966 (    )
16 INFORM:7 (119 391 119 )
17 INFORM:4967 (    )
```

Fig. 4.4. *System run representing 22 countries (1 university per country); partial report form the sniffer agent*

3. Adding more intelligent decision making components (MCDM modules), so that selections are based on a realistically selected set of criteria. We do not assume that our goal has to be to develop fully-functional modules, but rather establish which technology should be used to seamlessly integrate it into the system under development.
4. Making communication between agents more realistic by developing and/or utilizing existing ontologies and negotiation protocols concerning various aspects of "academic life".
5. Moreover, performing international tests (computers located in different countries) is compulsory as what we want to achieve is globally working system.

We will be reporting on our progress in the near future.

REFERENCES

[1] D. Handl, H.-J. Hoffmann, *Workflow agents in the document-centred communication in MALL2000 systems*, http://www.aois.org/99/handl.html

[2] A. Dogac, Y. Tambag, A. Tumer, M. Ezbiderli, N. Tatbul, N. Hamali, C. Icdem, and C. Beeri, *A Workflow System through Cooperating Agents for Control and Document Flow over the Internet* In: Cooperative Information Systems, 7th International Conference, CoopIS 2000

[3] M. Ganzha, W. Kuranowski, M. Paprzycki,*Socrates Agents — in Support o Student Mobility* In: Proceedings of the 17th Mountain Conference of the Polish Information Society, Szczyrk, Poland (to appear)

[4] M. Ganzha, W. Kuranowski, M. Paprzycki, S. Rahimi, M. Szymczak, *Designing an Agent-based Student Mobility Support System* In: In: W. Essaidi, N. Raissouni (ed.) Information and Communication Technologies International Symposium Proceedings, Al Khalij Al Arabi, Tetuan, Morocco, 2005, 44-50

[5] M. Gawinecki, Z. Vetulani, M. Gordon, M. Paprzycki, *Representing Users in a Travel Support System*, in: Proceedings of the ISDA 2005 Conference (to appear)

[6] M. Gordon, M. Paprzycki, *Designing Agent Based Travel Support System*, in: Proceedings of the ISPDC 2005 Conference, IEEE Computer Society Press, Los Alamitos, CA, 2005, 207-214

[7] *JADE. Java Agent Development Framework* See: `http://jade.cselt.it`

[8] N. R. Jennings, *An agent-based approach for building complex software systems*, Communications of the ACM, 44 (4), 2001, 35-41

[9] R. Krishnan , L. Munaga, K. Karlapalem, *XDoC-WFMS: A Framework for Document Centric Workflow Management System*, In: Data and Semantics of Web Information Systems Workshop (DASWIS) Japan 2001

[10] S. Lambert, et. al., *Knowledge Management for Organisationally Mobile Public Employees*, in: Wimmer M. A. (Ed.), KMGov 2003, LNAI 2645, 2003, 203-212

[11] S. Lambert, et. al., *A Framework for Experience Management in e-Government*, in: Proceedings of the 4th Eropean Conference on e-Goverment, 2004, pp. 451-460

[12] P. Maes, *Agents that Reduce Work and Information Overload*, Communications of the ACM, 37(7), 1994, 31-40

[13] Nguyen T.G., Hluchy L., Laclavik M., Balogh Z., Budinska I., Dang T.T. *Pellucid - Platform for Organisational Public Emploees.* International Conference on Emerging Telecommunication Technologies and Application - ICETA'2004, pp. , IEEE Computer Society, ISBN . September 2004, Kosice, Slovakia.

[14] Dang T.T, Hluchy L., Nguyen T.G., Budinska I., Laclavik M., Balogh Z *Knowledge Management and Data Classification in Pellucid* In: Intelligent Information Systems - IIS 2003: New Trends in Intelligent Information Processing and Web Mining, pp. 563-568, Springer-Verlag, Series: Advances in Soft Computing, ISBN 3-540-00843-8, June, 2003, Zakopane, Poland

[15] M. Laclavik, Z. Balogh, L. Hluchy, G. T. Nguyen, I. Budinska and T. T. Dang*Pellucid Agent Architecture for Administration Based Processes*In: International Conference on Intelligent Agents, Web Technology and Internet Commerce - IAWTIC 2003, in CDs. February 2003, Vienna, Austria.

[16] H. Nwana, D. Ndumu, *A perspective on software agents research*, The Knowledge Engineering Review, 14 (2), 1999, 1-18.

[17] *PS Tools* See: `http://www.sysinternals.com`

[18] Słota, R., Krawczyk, K., Dziewierz, M., Kitowski, J., Lambert, S., *Agent paradigm for accessing document repositories in Pellucid platform*, EuroWeb 2002 conference : the Web and the GRID: from e-science to e-business : Oxford 17–18 December 2002 / eds. Brian Matthews, Bob Hopgood, Michael Wilson. — Swindon : The British Computer Society, 2002. pp. 192–194.

Edited by: Shahram Rahimi, Raheel Ahmad
Received: October 11, 2005
Accepted: March 19, 2006

SCALABLE COMPUTING: PRACTICE AND EXPERIENCE
Volume 7, Number 1, pp. 59–75. http://www.scpe.org

ISSN 1895-1767
© 2006 SWPS

YOURSKYG: LARGE-SCALE ASTRONOMICAL IMAGE MOSAICKING ON THE INFORMATION POWER GRID

JOSEPH C. JACOB*, JAMES B. COLLIER , LORING G. CRAYMER , AND DAVID W. CURKENDALL

Abstract. The yourSkyG custom astronomical image mosaicking software has a web portal interface that allows custom access via ordinary desktop computers with low bandwidth network connections to high performance mosaicking software deployed on a computational grid, such as NASA's Information Power Grid (IPG). In this context, custom access refers to on-the-fly mosaicking to meet user-specified criteria for region of the sky to be mosaicked, data sets to be used, resolution, coordinate system, projection, data type and image format. The portal uses pipelines and data caches to construct multiple mosaics on the grid with high throughput.

Key words. Astronomy, virtual observatory, image, mosaic, web, grid, portal

1. Introduction.

In recent years the Astronomy community has witnessed rapid growth in the size and complexity of astronomical data sets due to rapid advances in remote sensing technology. The massive data sets that now exist collectively contain tens of terabytes of imagery and catalogs in wavelengths spanning the entire electromagnetic spectrum. Although this rich data store represents a significant opportunity for new scientific discoveries, it also represents a serious challenge to the community: How does one effectively and efficiently extract information from such a large and complex collection of data? The National Virtual Observatory (NVO) [1, 2, 3] is addressing this question in the United States and similar efforts exist elsewhere in the world [4, 5, 6, 7]. Many of the these virtual observatory projects are cooperating to ensure that they remain integrated and interoperable via the International Virtual Observatory Alliance (IVOA) [8].

As a community effort, the virtual observatory necessarily exhibits a loosely coupled, distributed architecture, with an emphasis on interoperability between components developed and deployed by domain experts in various areas. Since many of these components require an enormous amount of computation and data movement, the NVO needs to be deployed in a distributed, high performance, scalable computing environment. However, a significant fraction of astronomical research is conducted by scientists and students with limited resources, ordinary desktop computers and low bandwidth network connections. Therefore, to be effective the NVO also needs to provide portals to its high performance infrastructure that will make it usable by researchers anywhere.

1.1. Grid Computing in Astronomy.

The emergence of the virtual observatory concept coincides with the maturation of computational grids as a viable architecture for high-performance or data-intensive, distributed computations. The fundamental computational grid infrastructure includes both hardware-distributed, possibly heterogeneous processors interconnected by networks—and software to launch remote computations and to transport data to the processes that require them. The fundamental software infrastructure, provided by the Globus Toolkit, is what makes a collection of distributed computational resources into a functional computational grid, providing users with a single point of authentication for simultaneous access to all of the grid resources. In the grid development community, research is ongoing to bring into production more sophisticated grid software, layered on top of Globus, to provide additional functionality such as job monitoring, checkpointing, stop and restart, error recovery, planning, and scheduling.

The research described in this paper was conducted in 2002-2003, at which time the NVO was in its early stages and application of grid technology to astronomical research was very limited. At this time, a number of important web-based systems were instantiated, which serve as a model for the grid-based and web-based architectures prevalent today.

A number of projects used grid computing to allow science users around the world to access computational software over the Internet. The main advantage is that deploying these algorithms as grid and web services makes them accessible to science users with limited resources and only lightweight computers and network connections. One example of this is the Hera architecture [9], which makes it possible to run the High Energy Astrophysics Science Archive Research Center (HEASARC) data analysis software at NASA's Goddard Space Flight Center (GSFC) on a remote server via a simple graphical interface. Astrocomp [10] is a web portal

*Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, CA 91109-8099, Joseph.Jacob@jpl.nasa.gov for access to software for N-body simulations. Similarly, the yourSky web portal [11], described in this paper, provides remote access to JPL's parallel astronomical image mosaicking software via a simple web form interface.

The eSTAR Project [12] is an intelligent robotic telescope network that connects intelligent agents to telescopes and databases through grid and web services. This is an example of how grid computing is applied to science activities like Gamma-Ray Burst followup observations and the hunt for planets outside our solar system.

The Sloan Digital Sky Survey (SDSS) was an early adopter of web services technologies to facilitate access to large astronomical datasets. SDSS used many of the same technologies that are prevalent today in web-based commerce, including SOAP (Simple Object Access Protocol), XML (Extensible Markup Language), and WSDL (Web Service Description Language), to build services for resource discovery, data mining, visualization, and statistical analysis. The SkyQuery [13, 14] portal was implemented for SDSS using this web services architecture.

A number of projects related to virtual observatories are focused on using grid and web services technology to access multiple datasets (images and catalogs) from different archive centers and merge them to provide richer information content than is available from any of the datasets alone. This type of interoperability concept is demonstrated in the Aladin/GLU system [15], the European Space Agency (ESA) science archives [16], the Smithsonian Astrophysical Observatory (SAO) spectral archives [17], the Astrophysical Virtual Observatory (AVO) interoperability prototype [18], the On-Line Archive Science Information Services (OASIS) [19], and Grist (Grid Services for Astronomy) projects [20]. The Unified Content Descriptor (UCD) [21] prescribes a naming scheme for astronomical catalogs in order to facilitate this type of interoperability.

NASA's Information Power Grid (IPG) provided the grid infrastructure used in this work.[1] The IPG connected SGI Origin servers and Linux clusters distributed at NASA centers nationwide. The National Science Foundation (NSF) also sponsors a computational grid called the TeraGrid, which, at the time the research reported in this paper was conducted, linked together large Linux clusters at five sites, California Institute of Technology, San Diego Supercomputer Center, Argonne National Laboratory, National Center for Supercomputing Applications, and Pittsburgh Supercomputing Center. By September 2004, additional TeraGrid centers were added, including Indiana University, Oak Ridge National Laboratory, Purdue University, and Texas Advanced Computing Center, for an aggregate processing power of 40 teraflops, with 2 petabytes of disk storage, all interconnected with a 10-30 gigabits per second dedicated national network [22].

1.2. Image Mosaicking in Astronomy. The mosaicking described in this paper involves reprojecting input image plates to a common coordinate system, projection, equinox, and epoch, and combining the resulting plates to produce a single output image. There are strong science drivers for mosaicking. The most obvious is that large image mosaics enable analysis of celestial objects that either do not fit on a single image plate in the native image partitioning scheme used by a survey, or fall at the boundary between two or more neighboring plates. Also, mosaics enable analysis of the large-scale structure of the universe. In addition, mosaicking data sets in different wavelengths or from different surveys to the same coordinate grid enables multi-spectral analysis, which could be essential for identifying new, previously unknown, types of objects, or for identifying new objects that are so faint in a single wavelength that they are overlooked until combined with the signals from other wavelengths.

A number of software packages exist that can be used to construct astronomical image mosaics. This paper describes the yourSky software and its usage on the IPG. The yourSky software is the baseline for Montage [23, 24], a general science-grade astronomical image mosaicking toolkit that preserves both astrometry (object positions) and photometry (brightnesses) in the images. The Montage software was deployed as a service on the TeraGrid using a scheduler called Pegasus [25, 26] and Condor DAGMAN [27] to launch the computations on the grid in a manner that preserves all of the dependencies. Other notable astronomical mosaicking projects include SWarp [28] from the French TERAPIX center and MOPEX [29] from the Spitzer Science Center at Caltech.

1.3. From yourSky to yourSkyG. In this paper, we describe yourSky and yourSkyG, portals for high-performance, on-demand, astronomical image mosaicking. Both yourSky and yourSkyG can perform their high performance computations and data movement on conventional supercomputers, but yourSky requires use of a local multiprocessor system, while yourSkyG is capable of launching its computations on remote computers organized in a computational grid such as the IPG. A key characteristic of the portal architecture is that the

[1]NASA decommissioned the IPG in 2004.

data movements required to construct a requested mosaic and the actual location where the computations are carried out are transparent to the user who simply orders his mosaic by specifying the parameters that describe the mosaic. Regardless of where the computations are performed, these portals are accessible via lightweight client software—the ubiquitous web browser. The architecture of yourSkyG is motivated by the loosely-coupled, distributed nature of both the NVO and IPG infrastructures.

The yourSkyG portal is optimized for efficient processing of mosaic *ensembles*, multiple mosaic requests to be processed together. This mode of processing can dramatically improve throughput by (*i*) reducing the amount of data communication in cases where multiple mosaics require the same input image plates, and (*ii*) performing computation and communication for different mosaics in parallel where possible. Furthermore, we introduce the concept of *data reservoirs*, carefully managed data caches maintained at each stage of the data flow pipeline that have the effect of smoothing out variations in throughput as the availability of and load on shared grid resources change over time.

The architecture of yourSky is summarized in Section 2. Enhancements required to produce yourSkyG, running on the IPG, are described in Section 3. Optimizations for processing multiple mosaic requests are described in Section 4. Performance results are provided in Section 5. Finally, a summary is provided in Section 6.

2. The yourSky Portal.

The only client software required to use the yourSky custom astronomical image mosaic server is the ubiquitous web browser. By filling out and submitting the request form, users have custom access on their desktop to all of the publicly released data from the member surveys. In this context, "custom access" refers to new technology that enables on-the-fly astronomical image mosaicking to meet user-specified criteria for region of the sky to be mosaicked, data set to be used, resolution, coordinate system, projection, data type, and image format. All mosaic requests are fulfilled from the original archive data so that the domain experts maintain control and responsibility for their data and data corruption due to resampling is minimized because only one reprojection is done from the raw input data. Currently the data archives that are accessible with yourSky are the Digitized Palomar Observatory Sky Survey (DPOSS) [30] and the Two Micron All Sky Survey (2MASS) [31]. DPOSS has captured the entire northern Sky at 1 arc second resolution in three visible wavelengths. 2MASS has captured the entire sky at 1 arc second resolution in three infrared wavelengths. The yourSky architecture supports expansion to include other surveys, without regard to the native image partitioning scheme used by a particular survey.

2.1. Architecture.

The architecture for yourSky is illustrated in Fig. 2.1. In the figure, the numbered descriptions on some of the arrows give the steps taken to fulfill a typical mosaic request. The procedure is as follows. The clients at the top left of the illustration are the web browsers that may be used to submit requests to yourSky. A simple HTML form interface, shown in Fig. 2.2, is used to specify the parameters that are to be passed to the custom astronomical image mosaicking software. The mosaicking software and the mosaic parameters are described in detail in Section 2.2. The yourSky Mosaic Request Manager running on the yourSky server checks for mosaic requests and hands them off to the Mosaic Request Handler, using the user priority scheme described in Section 2.3. The Mosaic Request Handler queries the Plate Coverage Database, described in Section 2.4, to determine which input image plates from DPOSS or 2MASS are required to fulfill the mosaic request. A fixed size data cache is maintained on the yourSky server to store the input image plates required to build recent mosaic requests. If all of the required input image plates are already present locally in the data cache, the mosaic is constructed immediately using the custom astronomical image mosaicking software. If some of the required input image plates are not already cached locally, they need to be retrieved from their respective archives. Therefore an "archive request" is issued. The yourSky Archive Request Manager checks for archive requests and hands them off to the Archive Request Handler, which retrieves the required input image plates from the appropriate remote archive. Once all of the input image plates for a request have been cached on a local disk, the custom astronomical image mosaicking software is launched to construct the mosaic. When the mosaic, built precisely to match the user's request parameters, is ready an email is sent back to the user with the URL where the image mosaic can be downloaded.

2.2. Custom Astronomical Image Mosaicking Software.

The heart of the yourSky server is the custom astronomical image mosaicking software that is used to construct an image mosaic precisely matching user-specified parameters. The inputs to the mosaicking software are a list of input images to be mosaicked and the custom parameters that determine the properties of the mosaic to be constructed. The only requirements on the input images are the following. First, they must comply with the standard dictated by the Flexible Image

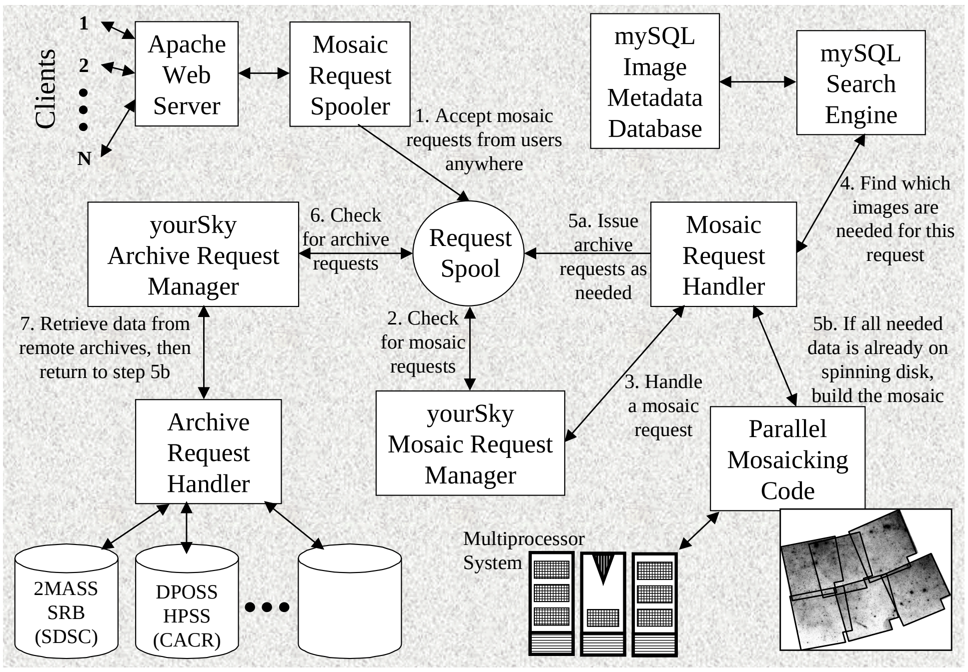
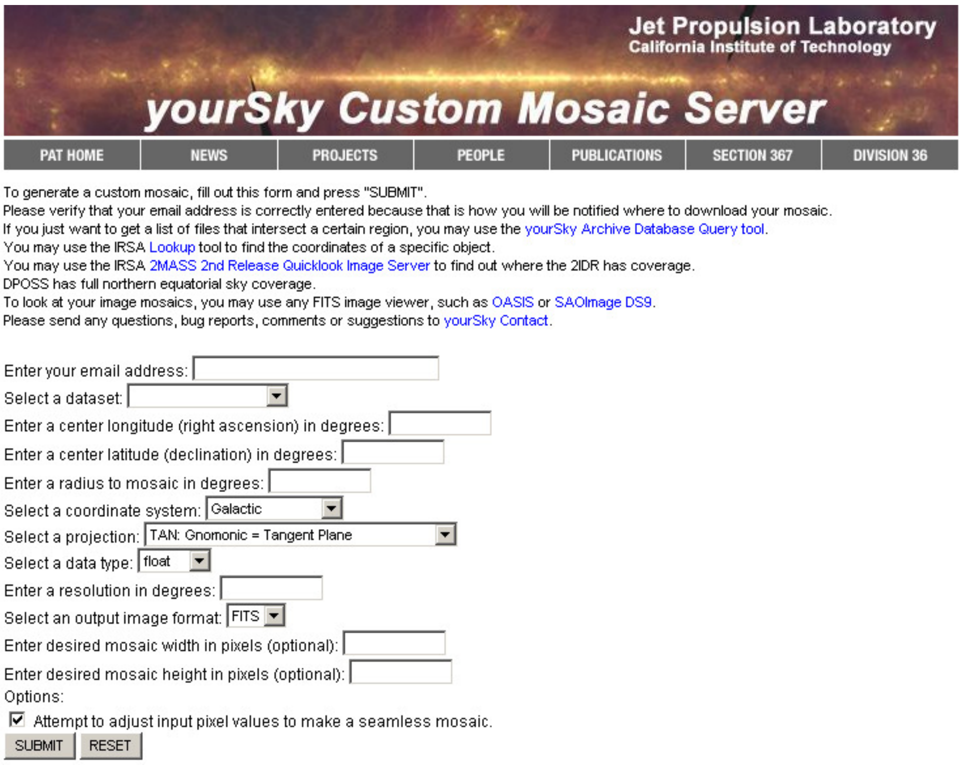

Clients
1
2
N
Apache Web Server
Mosaic Request Spooler
mySQL Image Metadata Database
mySQL Search Engine
1. Accept mosaic requests from users anywhere
4. Find which images are needed for this request
yourSky Archive Request Manager
6. Check for archive requests
Request Spool
5a. Issue archive requests as needed
Mosaic Request Handler
7. Retrieve data from remote archives, then return to step 5b
2. Check for mosaic requests
5b. If all needed data is already on spinning disk, build the mosaic
3. Handle a mosaic request
Archive Request Handler
yourSky Mosaic Request Manager
Parallel Mosaicking Code
Multiprocessor System
2MASS SRB (SDSC)
DPOSS HPSS (CACR)

Fig. 2.1. *The architecture of yourSky supports fully automated mosaicking, including retrieval of the input image plates from the remote survey archives.*


Jet Propulsion Laboratory
California Institute of Technology
yourSky Custom Mosaic Server
PAT HOME
NEWS
PROJECTS
PEOPLE
PUBLICATIONS
SECTION 367
DIVISION 36

To generate a custom mosaic, fill out this form and press "SUBMIT".
Please verify that your email address is correctly entered because that is how you will be notified where to download your mosaic.
If you just want to get a list of files that intersect a certain region, you may use the yourSky Archive Database Query tool.
You may use the IRSA Lookup tool to find the coordinates of a specific object.
You may use the IRSA 2MASS 2nd Release Quicklook Image Server to find out where the 2IDR has coverage.
DPOSS has full northern equatorial sky coverage.
To look at your image mosaics, you may use any FITS image viewer, such as OASIS or SAOImage DS9.
Please send any questions, bug reports, comments or suggestions to yourSky Contact.

Enter your email address:
Select a dataset:
Enter a center longitude (right ascension) in degrees:
Enter a center latitude (declination) in degrees:
Enter a radius to mosaic in degrees:
Select a coordinate system: Galactic
Select a projection: TAN: Gnomonic = Tangent Plane
Select a data type: float
Enter a resolution in degrees:
Select an output image format: FITS
Enter desired mosaic width in pixels (optional):
Enter desired mosaic height in pixels (optional):
Options:
☑ Attempt to adjust input pixel values to make a seamless mosaic.

Fig. 2.2. *The yourSky custom mosaic web form interface.*

Transport System (FITS), a data format that is well understood by the astronomy community and has long been used as the de facto method for sharing data within the community [32]. FITS format images encapsulate the image data with keyword-value pairs that give additional information about how the data values in the image map to locations on the sky. The second requirement for input images to the mosaicking software is that the FITS header must contain valid World Coordinate System (WCS) information. The WCS defines pixel-to-sky and sky-to-pixel coordinate transformations for a variety of coordinate systems and projections commonly used by the astronomy community [33].

2.2.1. Custom Access. With yourSky, the emphasis is on custom access to astronomical image mosaics. The following parameters may be used to specify the mosaic to be constructed:

1. Center right ascension and declination: Required parameters, analogous to the CRVAL1 and CRVAL2 FITS keywords, which specify the location on the celestial sphere of the tangent point for the image projection plane. By default, this center of projection is placed at the center pixel in the mosaic, analogous to the CRPIX1 and CRPIX2 FITS keywords.
2. Resolution: Required parameters, analogous to the CDELT1 and CDELT2 FITS keywords, which specify the pixel size in degrees in each of the two image dimensions at the mosaic center of projection.
3. Radius in degrees: Optional parameter that limits the mosaic size using degrees from the mosaic center. If not specified, the radius is determined automatically from the region of coverage of the input image plates.
4. Width and height in pixels: Optional parameters, analogous to the NAXIS1 and NAXIS2 FITS keywords, that limit the mosaic size using a specific number of pixels. These parameters supersede the radius in degrees if that is given as well. If not specified, the size is determined automatically from the region of coverage of the input image plates.
5. Coordinate system: Required parameter, analogous to the first half of the CTYPE1 and CTYPE2 FITS keyword values, that specifies the alignment of the mosaic axes in 3-D space. Four coordinate systems are supported: galactic, ecliptic, J2000 equatorial, and B1950 equatorial.
6. Projection: Required parameter, analogous to the second half of the CTYPE1 and CTYPE2 FITS keyword values, that specifies how locations on the celestial sphere are mapped to the image projection plane. All of the projections specified by WCS are supported: Linear (LIN), Gnomonic (TAN), Orthographic (SIN), Stereographic (STG), Zenithal/Azimuthal Perspective (AZP), Zenithal/Azimuthal Equidistant (ARC), Zenithal/Azimuthal Polynomial (ZPN), Zenithal/Azimuthal Equal Area (ZEA), Airy (AIR), Cylindrical Perspective (CYP), Cartesian (CAR), Mercator (MER), Cylindrical Equal Area (CEA), Conic Perspective (COP), Conic Equidistant (COD), Conic Equal Area (COE), Conic Orthomorphic (COO), Bonne (BON), Polyconic (PCO), Sanson-Flamsteed Sinusoidal (SFL), Parabolic (PAR), Hammer-Aitoff (AIT), Mollweide (MOL), COBE Quadrilateralized Spherical Cube (CSC), Quadrilateralized Spherical Cube (QSC), Tangential Spherical Cube (TSC), Digitized Sky Survey Plate Solution (DSS), and Plate fit polynomials (PLT).
7. Image Format: Required parameter that specifies the output mosaic image format (FITS is recommended). The following image formats are currently supported: FITS, JPEG, PGM, PNG, TIFF, and Raw Data.
8. Data Type: Required parameter that specifies the data type of the mosaic pixels. This is analogous to the BITPIX FITS keyword. The data types currently supported are 8-, 16-, and 32-bit signed and unsigned integer, and single and double precision floating point.
9. Quantization Extrema: Optional parameters that specify the minimum and maximum over which to stretch the input pixel values for those data types that require quantization to a limited number of output bits per pixel (especially, 8-bit and 16-bit integers). The user can specify these values to control how many gray levels in the output mosaic are assigned to low or high intensity regions of the sky.
10. Pixel Masks: Optional masks may be specified to discard pixels around the outer perimeter or from particular rectangular regions in each input image.
11. Background Matching: Logical parameter that specifies whether or not yourSky should attempt to match the background intensities among the input images that comprise a mosaic in an attempt to produce a mosaic that is as seamless as possible.

2.2.2. Parallel Mosaicking Algorithm. The yourSky mosaicking algorithm is designed to be able to handle arbitrarily sized mosaic requests from typical small requests covering a single celestial object to all-sky mosaics at full resolution. Also, the algorithm is efficient in the face of arbitrarily sized input image plates, so that yourSky can be extended to support other image archives without consideration of the native image partitioning scheme used by the archive. For example, the two surveys currently accessible by yourSky have drastically different native image partitioning schemes, from millions of small 2 MB image plates in the case of 2MASS to thousands of much larger 1 GB plates in the case of DPOSS. In addition, the mosaicking algorithm is designed to support arbitrary mappings from input image pixels to output mosaic pixels.

The mosaicking proceeds in two phases, Analysis and Build. During Analysis, the following is accomplished. First, the mosaic width and height are determined if they are not provided explicitly as part of the user-specified parameter set. Second, the pixel coordinates that intersect the mosaic are determined for each input image along with the corresponding intersection coordinates from the mosaic. These coordinates are used to set loop bounds and buffer sizes during the Build phase. Third, in cases where the data type requires quantization to a limited number of output bits per pixel in the output mosaic (e.g., 8-bit and 16-bit integers), the minimum and maximum over which the pixel values should be quantized are determined if these extrema are not specified explicitly as part of the user-specified parameter set. Fourth, if background matching is to be done, the intensity correction for each input image plate is determined.

During the Build phase the information gathered during Analysis is used to construct the custom mosaic. If the mosaic is to be lower resolution than the input image plates, the outer loop is over the input image pixels and the mosaic pixel values are calculated as the average of the input pixels for which the pixel center falls within the mosaic pixel region of coverage. If the mosaic is to be roughly the same or higher resolution than the input image plates, the outer loop is over the mosaic pixels and the mosaic pixel values are computed to be the result of sampling from the input images using bilinear interpolation. In either case, mapping from input pixel coordinates to output pixel coordinates is done by first mapping from input pixels to a location on the sky, then mapping from the sky coordinates to the output pixel coordinates, as illustrated in Fig. 2.3.

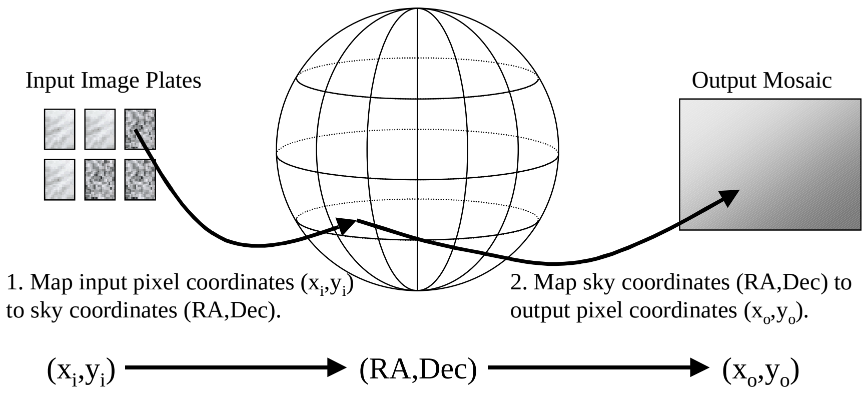

Input Image Plates
Output Mosaic
1. Map input pixel coordinates (x_i,y_i) to sky coordinates (RA,Dec).
2. Map sky coordinates (RA,Dec) to output pixel coordinates (x_o,y_o).
$(x_i,y_i) \longrightarrow (RA,Dec) \longrightarrow (x_o,y_o)$

Fig. 2.3. *Mapping from input pixel coordinates to output pixel coordinates is done in two steps. First, the input coordinates are mapped to a position on the sky, then that position on the sky is mapped to the output mosaic coordinates.*

The mosaicking proceeds in parallel during both Analysis and Build, with each processor being assigned a subset of the input image pixels. By default the input images are assigned to processors in a round robin fashion, with one processor per image, but the user can reconfigure this at run-time by specifying the number of processors to be assigned to each input image. Assigning multiple processors to each input image plate dramatically improves efficiency for archives, such as DPOSS, that have such large image plates that only a single or a few input image plates are required for a typical mosaic request. If multiple processors are assigned to each input image, a group synchronization among the processors assigned to the same image is required for each image so that Analysis results can be accumulated and shared. Also, in all cases, a global synchronization is required between the Analysis and Build phases so that Analysis results that relate to the entire mosaic, such as pixel value distributions required to calculate the appropriate quantization extrema, can be accumulated and shared. The software should be portable because it is written in ANSI C and all inter-processor communication and synchronization is done using Message Passing Interface (MPI), which has been implemented on many platforms [34].

2.2.3. Sample Mosaics. Some sample image mosaics, constructed with the yourSky custom image mosaicking software, are shown in Figs. 2.4, 2.5 and 2.6.

Fig. 2.4 shows a full 90 arc second resolution, all-sky, $14,400 \times 7,200$ pixel mosaic constructed from 430 Infrared Astronomical Satellite (IRAS) [35] image plates in each of 4 wavelengths, 12, 25, 60, and 100 μm. High performance exploration of this and other large data sets is possible using visualization software developed previously at JPL [36], [37].

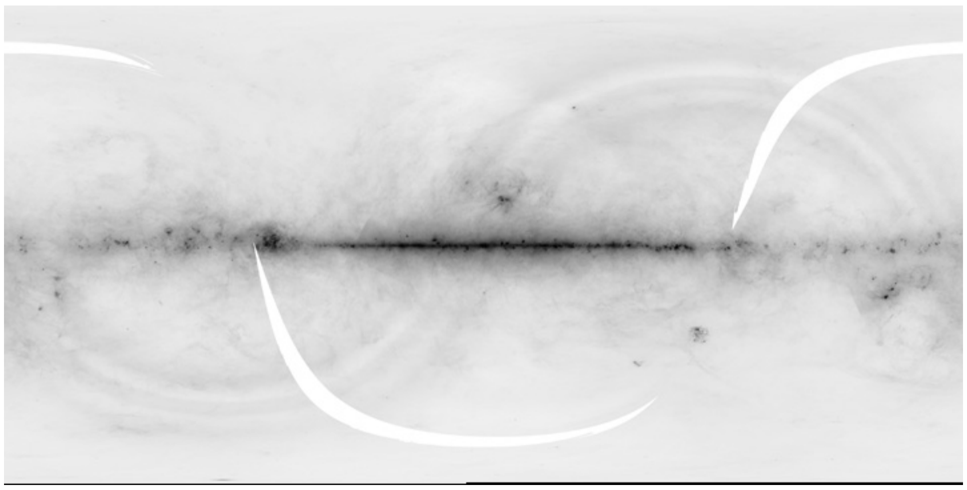


FIG. 2.4. *IRAS all-sky mosaic in the Cartesian (CAR) projection at 90 arc second resolution, constructed from 430 IRAS image plates in each of four wavelengths. The full resolution mosaic is* 14400×7200 *pixels.*

Fig. 2.5 shows a center of the galaxy mosaic from the 2MASS H band (1.65μm wavelength) before and after background matching is performed. The striped appearance without background matching is primarily due to atmospheric effects that become more pronounced as the path length through the atmosphere gets longer at different look angles. The background matching algorithm used by yourSky results in a more seamless mosaic, but edge effects are still visible. The Montage algorithms, described earlier, can be used to improve this background matching algorithm further.

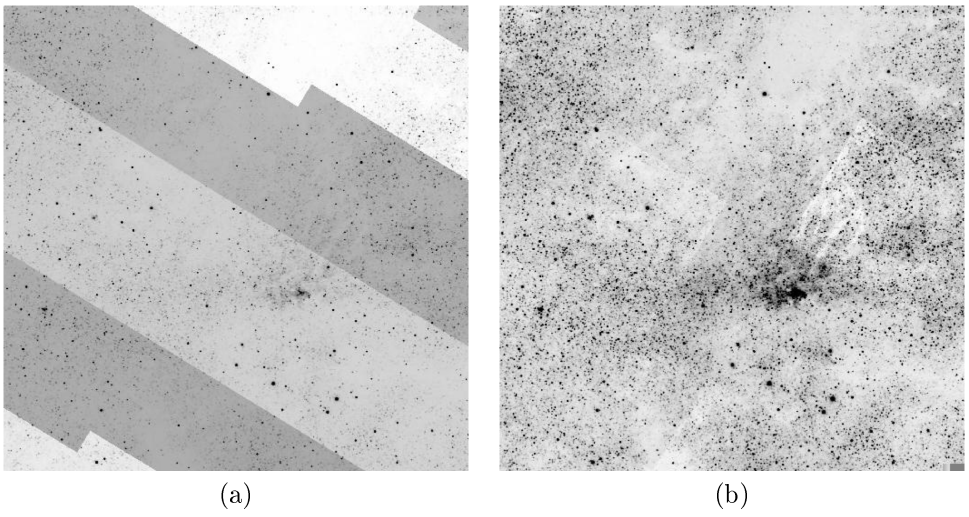


FIG. 2.5. *2MASS H band (*1.65μm *wavelength) center of the galaxy mosaic constructed from 16 2MASS image plates at 1 arc second resolution (a) without and (b) with background matching.*

Fig. 2.6 shows a DPOSS F band (650 nm wavelength) mosaic of M31 in a Galactic Tangent Plane projection. The mosaic shown in the figure is the center part of a larger $34,816 \times 36,352$ single precision floating point mosaic constructed from 9 DPOSS plates at full 1 arc second resolution.

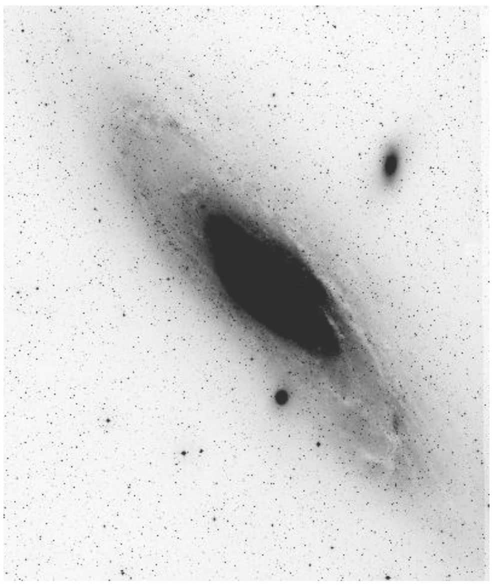


Fig. 2.6. *DPOSS F band (650 nm wavelength) mosaic of Andromeda (M31) at 1 arc second resolution. The image shown here is the center of a much larger mosaic constructed from 9 DPOSS plates.*

2.3. Request Management. Simultaneous mosaic requests are accepted from a simple HTML form submitted from the yourSky mosaic request web page, and queued on the yourSky server by a Common Gateway Interface (CGI) program interfacing with the Apache web server. The mosaic parameters for each request are stored on the yourSky server along with the identity of the user that submitted the request. The yourSky Mosaic Request Manager, shown in the architecture diagram in Fig. 2.1, needs to locate these mosaic requests and assign them one at a time to the Mosaic Request Handler. A user priority scheme is in place that starts off all users with equal priority. As requests are processed the user priorities change based on the number of mosaic pixels produced by each user in the past period referred to as the "priority window", currently set to 1 week. Users with the least number of mosaic pixels produced in the priority window period have highest priority for future mosaic requests. Furthermore, a mosaic request in progress that has had to wait for input image plate retrieval from a remote archive gets the highest priority to run next once all of the required input images have been retrieved. This ensures that all users get a chance to have their mosaic constructed and no single user will dominate all the available resources.

2.4. Plate Coverage Database. In order to be accessible by yourSky, all member surveys have to be included in the Plate Coverage Database that contains the minima and maxima of the longitudes (right ascensions) and latitudes (declinations) in each of the supported coordinate systems for all of the input image plates. The yourSky Mosaic Request Handler queries this database to determine which input image plates are needed to fulfill each mosaic request. The open source database, MySQL, is used to store this plate coverage information [38], [39]. The result of the query to the Plate Coverage Database is a list of the input image plates that are required to fulfill the mosaic request. These input image plates are retrieved from the appropriate remote archives, staged in a local data cache, and provided as an input to the image mosaicking software described in Section 2.2. The plate coverage database is also available as a stand-alone service, called the yourSky Archive Database Query.

2.5. Data Management. A data management scheme is implemented on the yourSky server to manage both a data cache for the input image plates, used to fulfill recent mosaic requests, and a work area, used to store recently constructed mosaics until they are downloaded.

The input data cache is maintained at a fixed size with image plates discarded on a least recently used basis. This enables mosaics to be recomputed with some changes to the custom request parameters without having to repeat the input image plate retrieval from the remote archives if the new request is resubmitted before the input images are purged from the cache. Also, mosaics of popular regions of the sky are likely to have their input image plates already cached on the yourSky server from previous requests, so they can be constructed more quickly.

Mosaics that have been completed are stored in a work area from which they may be downloaded by the appropriate users. Currently mosaics are purged after a 1 week period expires.

2.6. Data Archive Access. All of the publicly released data from two archives are currently accessible by yourSky, the Digitized Palomar Observatory Sky Survey (DPOSS) and the Two Micron All Sky Survey (2MASS) Second Incremental Data Release (2IDR).

DPOSS has captured nearly the entire northern sky at 1 arc second resolution in three wavelengths, 480 nm (J Band—blue), 650 nm (F Band—red), and 850 nm (N Band—near-infrared). The survey data were captured on photographic plates by the 48-inch Oschin Telescope at the Palomar Observatory in California [30]. The total size of the DPOSS data accessible by yourSky is roughly 3 TB, stored in over 2,600 overlapping image plates on the High Performance Storage System (HPSS) [40] at the Center for Advanced Computing Research (CACR) at the California Institute of Technology. The DPOSS plates are each about 1 GB in size and contain $23,552 \times 23,552$ pixels covering a roughly 6.5×6.5 degree region of the sky. The yourSky server uses a client program called the Hierarchical Storage Interface (HSI) to retrieve selected DPOSS plates in batch mode from the HPSS [41].

2MASS has captured nearly the entire sky at 1 arc second resolution in three near-infrared wavelengths, 1.25μm (J Band), 1.65μm (H Band), and 2.17μm (K_S Band). The survey data were captured using two 1.3 meter telescopes, one at Mt. Hopkins, AZ and one at the Cerro Tololo Inter-American Observatory (CTIO) in Chile [31]. The 2MASS archives contain roughly 10 TB of images and the subset that was released as part of the 2MASS Second Incremental Release (2IDR), nearly 4 TB, is fully accessible by yourSky. This 4 TB of data is stored in about 1.8 million overlapping plates managed by the Storage Resource Broker (SRB) at the San Diego Supercomputer Center (SDSC). Each 2MASS plate is about 2 MB in size and contains $512 \times 1,024$ pixels covering a roughly 0.15×0.30 degree region of the sky. The SRB is a scalable client-server system that provides a uniform interface for connecting to heterogeneous data resources, transparently manages replicas of data collections, and organizes data into “containers” for efficient access [42]. The yourSky server uses a set of client programs called SRB Tools to access selected 2MASS plates in batch mode from the SRB.

3. From yourSky to yourSkyG. In Section 2 we described how yourSky enables custom astronomical mosaic construction on a local multiprocessor system. Here, “local” refers to the fact that the mosaic computations are performed on the same machine that hosts the web server. The objectives of yourSkyG are (*i*) to provide the same kind of custom, web-accessible mosaic service as yourSky, but to perform the compute intensive portions remotely on a grid, and (*ii*) to make much larger mosaicking jobs feasible by leveraging the computational power of the grid to full advantage.

The yourSkyG portal runs on a local grid portal system and maintains compliance with grid security. It initiates data transfers to and from, and job executions on, a remote IPG computer. Local security rules dictate that the web services offered by yourSkyG and the grid portal may not be hosted on the same system. Instead, the web interface and Mosaic Request Manager are resident on a local web portal system that has no direct connection to the grid, as illustrated in Fig. 3.1. This web portal system accepts user requests and stores them in files on a local disk that is also accessible by the grid portal system. The yourSkyG job manager with IPG authentication is hosted on the grid portal system. Periodically this job manager checks for mosaic requests. When a request is found, the job manager does the following:

1. Retrieve the required input images from the remote sky survey archives
2. Get a grid proxy in order to authorize grid access
3. Upload the required input images to the target grid system
4. Generate a file in the Globus Resource Specification Language (RSL) that specifies the job to be run on the grid
5. Execute this RSL and wait for the resulting remote grid job to finish
6. Download the resulting output image
7. Remove files that are no longer needed from the remote grid system
8. Notify the user via e-mail
9. Optionally store the mosaic in an SRB archive

This architectural change successfully separates the grid portal from the less secure web portal, without requiring significant modifications to the yourSky architecture. However, this does requires coordination between the web and grid portals, which is accomplished through files stored on the shared disk.

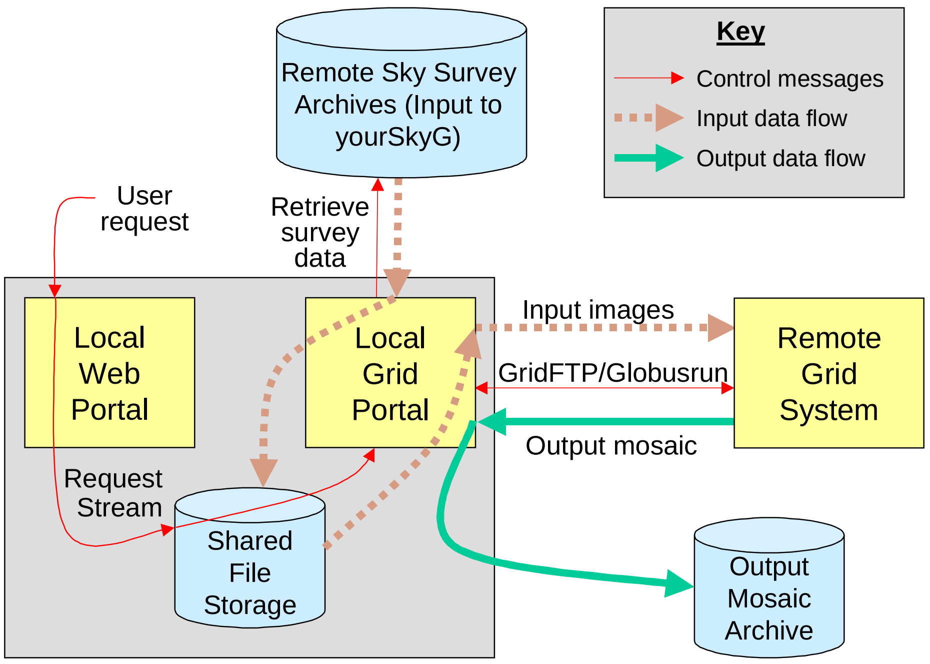


Fig. 3.1. *In the yourSkyG architecture, there is a clear separation between the local web and grid portals and the remote grid compute system where the mosaic processing is performed. The local grid portal coordinates the data flow and remote job execution.*

4. Optimizations for Large-Scale Mosaic Ensembles. The grid offers a wealth of computational, storage, and networking resources together with a rich set of tools for accessing them. The central problem is how to manage all of these resources in order to achieve an acceptable sustained throughput rate. This resource management problem is not only complex but also dynamic, with resource availability and usage changing over time. Resource management is of particular importance for the problem addressed in this paper, astronomical image mosaicking, because this is not only a compute intensive task but also a data intensive task. Not surprisingly, both computational resource scheduling and data communication are important issues. The architecture of yourSkyG addresses this resource management problem in the context of processing multiple mosaics, i. e., 100 or more mosaics at a time.

The key architectural features of yourSkyG are (*i*) state-based data flow, (*ii*) pipeline processing, and (*iii*) data reservoirs, described below. Together these produce beneficial characteristics in yourSkyG such as controlled usage of shared grid resources, improved throughput, and a degree of fault tolerance. Throughput is improved as a result of overlapping computation and communication, caching data close to the processing, and careful ordering of the mosaics to be processed to maximize the use of the data caches and minimize the amount of communication required. The use of a processing pipeline means that many different mosaics may be processed at the same time but at different phases in the processing sequence. For example, while one or more mosaics are being computed on the grid, the output from previous mosaic computations can also be downloaded and the inputs for later mosaics can be uploaded.

The following provides more detail about the key architectural features of yourSkyG, as well as some performance results summarizing our experiences in creating a set of 900 overlapping mosaics from DPOSS, collectively covering the entire northern sky in 6×6 degree patches with 1 arc second sampling. This ensemble of mosaics, totaling about 1.7 terabytes per wavelength, can be visualized using an all-sky, web-based image browsing service [43].

4.1. Ensemble Request Management. A mosaic ensemble request is a set of parameters, represented as a set of keyword-value pairs, that describe the mosaics to be constructed. Some of these parameters apply to all of the mosaics in the ensemble, for example, input survey and wavelength, coordinate system, projection, resolution, width and height in pixels. Others specify the individual mosaics, for example, right ascension and declination of the center.

Parameters are added to the mosaic ensemble request in two steps. First, the parameters that specify the input images required to compute each mosaic are added. Second, the parameters that specify the grid resources to be used in this computation are added. The final parameter set contains all the information required for computing the mosaics.

4.2. Ordering Mosaic Computations for Efficient Data Transport. Since large files must be moved to and from the remote grid system, data transport efficiency is extremely important. In our DPOSS plate ensemble example, each output DPOSS mosaic is 1.9 GB in size and on average requires processing of 7.6 input image plates each 1.1 GB for an average total transfer of over 10 GB. We have found in this example that without an adequate data transport strategy data transfer time can easily dwarf mosaic computation time.

If the requested mosaics in an ensemble are localized on the sky, some input images may be required for multiple output mosaics. The yourSkyG portal takes advantage of this by maintaining a data cache of input images on the target grid system so that these images may be reused whenever possible to reduce the volume of data transferred. Moreover, the order in which the mosaics in an ensemble are processed is selected to take maximum advantage of the available data cache. Returning to our DPOSS plate ensemble example, a single mosaic processed by itself requires on average the transfer of 8.5 GB of input images. However, if an ensemble of these mosaics are processed together, taking advantage of computation reordering and data caching may eliminate up to 85% of the input data transfers.

4.3. State-Based Processing Model. Conceptually, the processing of a mosaic job is modeled as a sequence of states and state transitions, as illustrated in Fig. 4.1. This model has been implemented so that each state is a directory and the objects in these states are files. Each state transition is then the movement of some file (object) from its current directory (current state) to a new directory (new state). The file that is moved from state to state may be an input image, an output mosaic, a mosaic job description, or a message of some kind. At any given time each possible state may be occupied by multiple objects.

An input image moves between the states of `input_awaiting_download`, `input_awaiting_upload`, and `input_cached`. The `input_awaiting_download` state means that the input image is scheduled for download from the remote survey archive. The `input_awaiting_upload` state means that the image is scheduled for upload to the remote grid system. The `input_cached` state means that the input image is preserved in a two-level cache, the primary on the remote grid system and the secondary on the local grid portal. The secondary cache is used to correct data transfer errors to the primary cache and for reuse with later mosaic ensemble requests. An output image mosaic moves between the states of `output_computed`, `output_downloaded`, `output_archived`, `output_purged_from_grid`, and `output_cached`. The transition to the `output_purged_from_grid` state means that files on the grid that are no longer needed have been removed to free grid resources for later jobs.

A mosaic ensemble request is partitioned into subsets, each of which is treated as a single batch job on the grid, referred to here as a "job". Each job moves between the states of `job_identifying_inputs_for_upload`, `job_awaiting_input_upload`, `job_ready_to_submit`, `job_queued`, `job_executing`, and `job_completed`.

Upon job completion, a message is sent back to the grid portal system that specifies the output files to be downloaded, their locations on the remote grid system, and their sizes for automated error checking.

4.4. Data Flow Model Using Concurrent Asynchronous Processes. In the yourSkyG data flow model, each state transition is implemented as a separate process and all of these processes execute asynchronously. Each state transition process executes in its own `current_state` directory, reads each file in that directory in time order, applies some operation to that file, and moves it to a `next_state` directory.

There is no direct communication between these processes and no centralized control. A state transition process executes whenever there is data available and processing occurs as rapidly as local resources allow. An output from a state transition process becomes an input for some other state transition process. The result is an efficient data flow architecture.

This design has beneficial software engineering features. Each state transition process is a small module easily constructed and modified, and easily inserted or removed from a much larger software structure. Most of the functions of a state transition process are common among all of these processes and need only be implemented once and reused.

This fine-grained architecture provides the user a high degree of control over the executing system. For example, one process that appears to have a problem may be selectively halted for further study while the rest of the system continues to execute. Halting a process has no serious consequences other than the accumulation

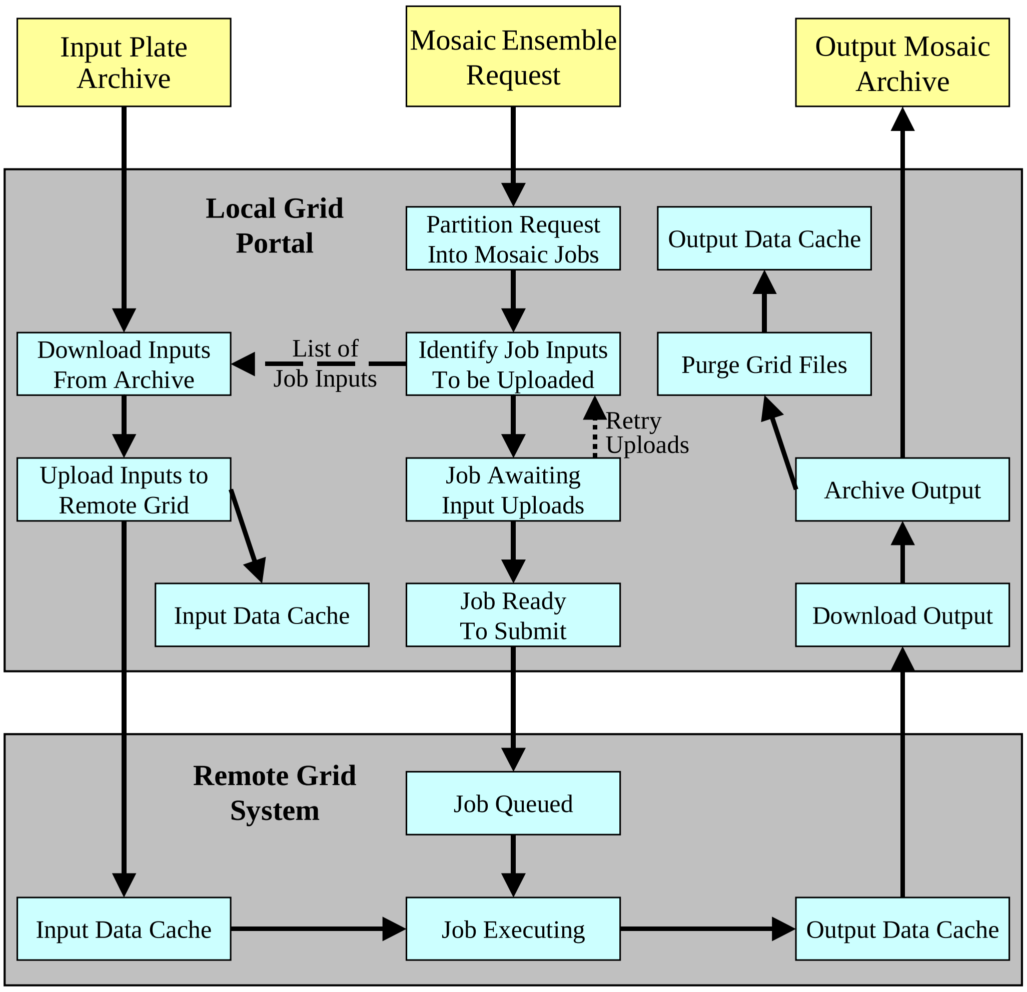


FIG. 4.1. *The yourSkyG computations to process multiple mosaics at a time on the grid can be modeled as a series of states and state transitions.*

of requests just prior to the inactive state transition. When the inactive process is restarted, processing returns to normal.

4.5. Pipeline with Reservoirs. The global architecture of yourSkyG is a pipeline created from a sequence of many independently executing state transition processes. Files move through this pipeline accumulating in different "reservoirs" and, in so doing, even out temporary fluctuations in local throughput. Given the data flow architecture, it is a straightforward extension to replicate the yourSkyG pipeline into multiple pipelines, executing asynchronously, and each targeting a different remote grid system.

A pipeline architecture by itself has the potential for greatly increased throughput. However, each of these state directories is also a data cache where files can accumulate until processing resources are available. Conceptually, each of these is a reservoir. Files accumulate in a reservoir when the state transition process following it is slower than the rest of the system and then drain out again when this state transition process speeds up again. As different resources speed up and slow down, bottlenecks move around but the reservoirs smooth out temporary variations and maintain a higher throughput rate. Only when the capacity of a resource is exceeded does that part of the pipeline shut down. For example, mosaic jobs that are ready to execute accumulate in the `job_ready_to_submit` directory but are only submitted to the batch queue when the number of mosaic jobs queued or executing on the remote grid system falls below a specific value. This prevents flooding the remote batch queue which would in turn block any other yourSkyG state process from accessing this resource; in particular, the state `job_awaiting_input_upload` would be unable to query for the current contents of the remote input data cache and would stall. However, even when one part of the pipeline shuts down for some reason, other parts do not. If the remote batch queue is full, the transfer of input files from archive to remote grid system will continue.

4.6. System Monitoring. We have found it essential to be able to monitor the functioning of the yourSkyG system in order to understand how the resources interact, determine the current system performance, and identify operational problems or implementation issues. The yourSkyG architecture makes this monitoring relatively simple.

Listing the contents of all the state directories provides a quick look at the current state of the pipeline. A bottleneck in the system is easily identified by an accumulation of files in a state directory and the directory identifies the particular resource to examine further. Inspection may include several computing systems and batch queues but these requests are easily automated.

In order to recover how the pipeline arrived in its current state, each process writes a log file to its `current_state` directory. Each log file records the name of each file processed, when it was processed, and the state transition applied. Redirecting `stdout` and `stderr` to the log file also captures any program or system error messages. Log file generation is one of the yourSkyG reusable components for state transition processes.

Generally, these few simple monitoring techniques will identify both the location and cause of a problem. If a particular problem is rare, a manual correction may be adequate. If a problem is chronic, then some form of automated error detection and correction may be necessary.

4.7. Automated Error Handling. We classify error handling into three sub-categories: detection, management, and correction.

Error handling begins with detection. In general, the supporting software used, such as the Globus tools, will report an error in a way that is automatically detectable by the calling program, usually a return value that lies within a specific range. However, there remain significant errors that are not reported this way. For example, a file transfer may fail at either end of the transfer route if one of the two host systems crashes or it may fail somewhere in the middle if a router malfunctions. The file may be nonexistent, zero length, or truncated at the destination, and that error not reported. The yourSkyG system performs an independent check on a transferred file to determine not only that the transferred file exists in its target location but also that that file has arrived having the correct size. For example, successful completion of a mosaic computation on a remote grid system is reported by the transfer of a small message file back to the local yourSkyG system containing a list of the output files to be downloaded and the file sizes that should be expected.

After an error has been detected, it must be managed in some way that protects the rest of the system so that validity of output data is not corrupted but regular processing continues with minimal disruption. This has been modeled and implemented as a state change; however, the new state cannot be the usual next state since that would indicate success. The preferred solution is to move the file responsible for this error to some preceding state and have the reprocessing correct the situation; however, careful design is required to prevent the possibility of an infinite loop. The more common alternative has been to create a fail state as a subdirectory of the state during which the error was detected and the file responsible for the error moved into this fail state directory. For example, if the transfer of a required input data file to the remote grid system fails, then that input data file is moved to the fail subdirectory of the current state directory, `input_awaiting_upload`, and the next input data file can be processed. Error correction is left for some other process.

Automatic error correction has been added to maintain reasonable processing throughput. For example, the `job_identifying_inputs_for_upload` state will check for a required input data file in the fail subdirectory for the `input_awaiting_upload` state before it attempts to download it again from the remote archive and will move this input data file back to the `input_awaiting_upload` state for another try. The function of the `job_awaiting_input_upload` state is to hold a mosaic job until it is verified that all necessary input files have been successfully transferred to the input data cache on the target grid system. Only then will the job be moved to the `job_ready_to_submit` state, meaning ready to submit to a batch queue on the target grid system. The transfer of input files is performed asynchronously by another pipeline and most mosaic job files will need to wait for these transfers to complete. However, if the `job_awaiting_input_upload` process finds that there is a required input file that not only is missing from the input data cache on the target grid system but also is missing from the `input_awaiting_upload` directory, then some error has occurred. The automated correction is to return the mosaic job file to the `job_identifying_inputs_for_upload` state for another try.

5. Performance Results. Here, we provide performance results when constructing a single mosaic on a multiprocessor system, as well as performance when constructing many of these mosaics in a batch processing mode on the grid.

5.1. Performance on a Single Mosaic. In this section we show timing results for the yourSky mosaicking software running on a SGI Origin 2000 with 300 MHz R12000 processors and 512 MB of RAM per processor. The first test mosaic is a 2 $\times$ 2 degree 2MASS mosaic of the galactic center at full one arc second resolution. The resulting mosaic is $7,201 \times 7,201$ single precision floating point pixels (207 MB) in size in a Galactic Tangent Plane projection, constructed from 174 2MASS plates totaling 365 MB in size. The second test mosaic is a 2 $\times$ 2 degree DPOSS mosaic of M31 at 10 arc seconds resolution, resulting in a 721 $\times$ 721 single precision floating point mosaic in the Galactic Tangent Plane projection. This 2.1 MB mosaic was constructed from 2 DPOSS plates (total size 2.2 GB). No background matching was performed for this test. Fig. 5.1 shows the wall clock time required to construct the mosaics on different numbers of processors on the Origin 2000. For the 2MASS mosaic, one processor was assigned to each input image plate, but all the processors were assigned to each input image plate for DPOSS. The plot shows the scaling curves for up to 64 processors.

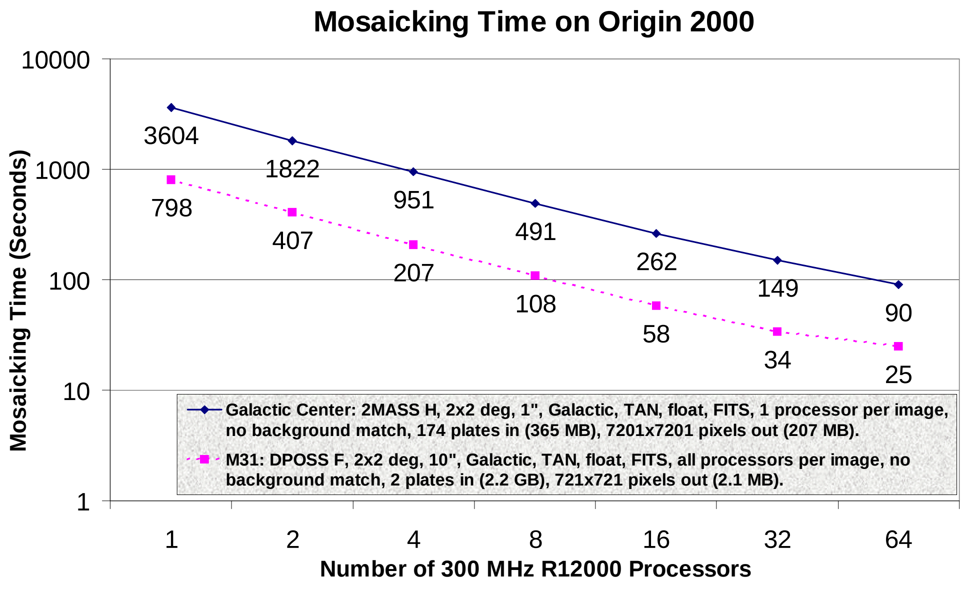


Fig. 5.1. *Mosaicking software performance on a SGI Origin 2000 for two different mosaic parameters, a 2 $\times$ 2 degree 1 arc second resolution mosaic of the galactic center in 2MASS with one processor assigned to each input image plate and a 2 $\times$ 2 degree 10 arc second resolution mosaic of M31 in DPOSS with all processors assigned to each input image plate.*

5.2. Performance on Multiple Mosaics. In this section we show typical throughput performance for yourSkyG generating an ensemble of 110 DPOSS mosaics. Each requested output mosaic is a 6×6 degree projection at 1 arc second resolution, resulting in 1.9 gigabytes per mosaic with 4 byte single precision floating point pixels. This yields a total size output of 205 gigabytes for the entire ensemble, which covers about 12% of the northern hemisphere covered by DPOSS. The input image plates are each 6.5×6.5 degrees at 1 arc second resolution, resulting in 1.1 gigabytes per input plate with 2 byte integer pixels. This yields a total size input of about 122 gigabytes for the entire ensemble.

The processing was automatically partitioned into 11 separate batch jobs, each computing 10 mosaics. The plot in Fig. 5.2 shows at any given time the percentages associated for each of the following values: total number of jobs started (on the local system), total number of input files transferred to the remote grid system cache, total number of jobs submitted to the batch queue on the remote grid system, and total number of output files transferred back to the local system. These values were extracted from the yourSkyG log files.

In order to ensure that valuable computational resources could be used for other purposes during data transfer, a 10-mosaic batch job was not scheduled until all of the input image plates it requires were transferred to a local disk on the remote grid system. The time from the start of the first job on the local grid portal to the return of the first output mosaic file is the time required to initialize the yourSkyG pipeline, which in this case

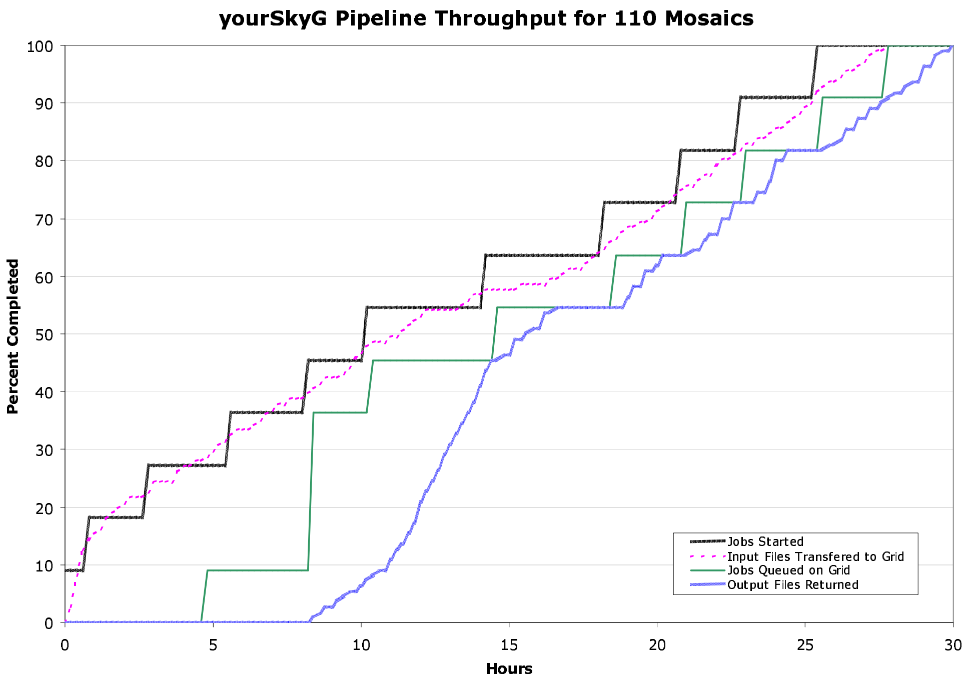


FIG. 5.2. *yourSkyG pipeline throughput for 110* 6×6 *degree DPOSS mosaics totalling about 205 GB in size.*

was 8 hours. The first half of this initialization time was spent transferring the input data required for the first 10-mosaic batch job and most of the second half was spent waiting in a batch queue on the remote grid system. However, once the pipeline was initialized, it returned computed mosaics at an average rate of 5 per hour or 120 per day. This is equivalent to a capability of mosaicking the entire DPOSS data set in about a week and the whole sky (both hemispheres) in roughly 2 weeks per wavelength at 1 arc second resolution.

This experiment also exercised the fault tolerance of the yourSkyG pipeline. Our logs indicate that during the computation of these 110 mosaics there were four failed input file transfers but each of these failures was automatically detected and corrected without user intervention. This fault tolerance is an essential feature for large scale processing on grids with distributed data archives and computational resources.

6. Summary. The yourSkyG portal uses the full computational power of the NASA Information Power Grid (IPG) to enable high-performance desktop access to custom astronomical image mosaics. The architecture of the portal allows it to exploit grid computing infrastructure, supercomputers and high bandwidth networks, on the server side. However, at the same time it is widely usable from virtually anywhere because the architecture also supports very lightweight computing resources on the client side, e.g., ordinary desktop computers with low bandwidth network connections. Since the user interface is a simple web form, the only client software required is the ubiquitous web browser, which most of the potential users probably already have and know how to use. This combination of being deployed in a high performance computing and communications environment while allowing access through simple portals running on the desktop makes yourSkyG a good match for the loosely coupled, distributed architecture of both the National Virtual Observatory (NVO) and the IPG.

The portal includes subsystems for: (i) construction of the image mosaics on multiprocessor systems or computational grids, (ii) managing simultaneous user requests, (iii) determining which image plates from member surveys are required to fulfill a given request, (iv) caching input image plates and the output mosaics between requests, and (v) retrieving input image plates from remote archives. The parallel image mosaicking software emphasizes custom access to mosaics, allowing the user to specify parameters that describe the mosaic to be built, including data sets to be used, location on the sky, size of the mosaic, resolution, coordinate system, projection, data type, and image format.

The grid work flow is optimized to achieve high-throughput processing of multiple mosaics to be constructed together as ensembles. The key architectural features for this mode of processing are a state-based data flow system, pipeline processing to overlap mosaic computations and data communications where possible, and the use of data reservoirs at various stages of the processing pipeline to provide a level of robustness in the face of varying load conditions on shared grid resources.

Acknowledgments. The yourSky portal was initially developed under sponsorship of NASA's Space Science Applications of Information Technology Program, Office of Space Science. The yourSkyG portal, a port of yourSky to the Information Power Grid, was sponsored by NASA's Computing, Networking, and Information Systems (CNIS) Program.

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

REFERENCES

[1] The National Virtual Observatory (USA), `http://us-vo.org`.

[2] National Virtual Observatory Science Definition Team Report: *Towards the National Virtual Observatory*, April 2002.

[3] NVO White Paper: *Toward a National Virtual Observatory: Science Goals, Technical Challenges, and Implementation Plan*, in Virtual Observatories of the Future, R. J. Brunner, S. G. Djorgovski, and A. S. Szalay, eds., Astronomical Society of the Pacific Conference Series, Vol. 225, pp. 353-372, 2001.

[4] R. G. Mann, A. Lawrence, C. Davenhall, R. McMahon, M. Irwin, N. Walton, G. Rixon, M. Watson, J. Osborne, C. Page, P. Allan, D. Giaretta, C. Perry, D. Pike, J. Sherman, F. Murtagh, L. Harra, B. Bentley, K. Mason, and S. Garrington, *AstroGrid: the UK's Virtual Observatory Initiative*, Proceedings of Astronomical Data Analysis Software and Systems XI, ASP Conference Series, Vol. 281, 2002, D. A. Bohlender, D. Durand, and T. H. Handley, eds.

[5] P. Quinn, P. Benvenuti, P. Diamond, F. Genova, A. Lawrence, and Y. Mellier, *The AVO Project: A European VO Initiative*, Proceedings of Astronomical Data Analysis Software and Systems XI, ASP Conference Series, Vol. 281, 2002, D. A. Bohlender, D. Durand, and T. H. Handley, eds.

[6] The Australian Virtual Observatory, `http://www.aus-vo.org/`.

[7] The Virtual Observatory - India, `http://vo.iucaa.ernet.in/~voi/`.

[8] The International Virtual Observatory Alliance (IVOA), `http://www.ivoa.net`.

[9] W. D. Pence, T. McGlynn, P. Chai, and C. Heikkila, *Hera: The HEASARC Web Based Data Analysis Environment*, Proceedings of SPIE Astronomical Telescopes and Instrumentation: Virtual Observatories Conference, August 2002.

[10] P. Di Matteo, R. Capuzzo Dolcetta, P. Miocchi, V. Antonuccio-Delogu, U. Becciani, A. Costa, and V. Rosato, *Astrocomp: A Web Portal for High Performance Computing on a Grid of Supercomputers*, Proceedings of Astronomical Data Analysis Software and Systems XII, ASP Conference Series, Vol. 295, 2003, H. E. Payne, R. I. Jedrzejewski, and R. N. Hook, eds.

[11] J. C. Jacob, R. J. Brunner, D. Curkendall, S. G. Djorgovski, J. C. Good, L. Husman, G. Kremenek, and A. Mahabal, *yourSky: Rapid Desktop Access to Custom Astronomical Image Mosaics*, Proceedings of SPIE Astronomical Telescopes and Instrumentation: Virtual Observatories Conference, August 2002.

[12] A. Allen, T. Naylor, I. Steele, D. Carter, J. Etherton, and C. Motteram, *eSTAR: Building an Observational GRID*, Proceedings of Astronomical Data Analysis Software and Systems XII, ASP Conference Series, Vol. 295, 2003, H. E. Payne, R. I. Jedrzejewski, and R. N. Hook, eds.

[13] A. Szalay, T. Budavari, T. Malik, J. Gray, and A. Thakar, *Web Services for the Virtual Observatory*, Proceedings of SPIE Astronomical Telescopes and Instrumentation: Virtual Observatories Conference, August 2002.

[14] T. Budavari, T. Malik, A. Szalay, A. Thakar, and J. Gray, *SkyQuery—A Prototype Distributed Query Web Service for the Virtual Observatory*, Proceedings of Astronomical Data Analysis Software and Systems XII, ASP Conference Series, Vol. 295, 2003, H. E. Payne, R. I. Jedrzejewski, and R. N. Hook, eds.

[15] P. Fernique, A. Schaaff, F. Bonnarel, and T. Boch, *A Bit of GLUe for the VO: Aladin Experience*, Proceedings of Astronomical Data Analysis Software and Systems XII, ASP Conference Series, Vol. 295, 2003, H. E. Payne, R. I. Jedrzejewski, and R. N. Hook, eds.

[16] C. Arviset, J. Dowson, J. Hernandez, P. Osuna, and A. Venet, *Interoperability of ESA Science Archives*, Proceedings of Astronomical Data Analysis Software and Systems XII, ASP Conference Series, Vol. 295, 2003, H. E. Payne, R. I. Jedrzejewski, and R. N. Hook, eds.

[17] D. J. Mink, and M. J. Kurtz, *Federating Catalogs and Interfacing Them with Archives: A VO Prototype*, Proceedings of Astronomical Data Analysis Software and Systems XII, ASP Conference Series, Vol. 295, 2003, H. E. Payne, R. I. Jedrzejewski, and R. N. Hook, eds.

[18] M. G. Allen, F. Genova, F. Ochsenbein, S. Derriere, C. Arviset, P. Didelon, M. Dolensky, S. Garrington, R. Mann, A. Micol, A. Richards, G. Rixon, and A. Wicenec, *Toward an AVO Interoperability Prototype*, Proceedings of Astronomical Data Analysis Software and Systems XII, ASP Conference Series, Vol. 295, 2003, H. E. Payne, R. I. Jedrzejewski, and R. N. Hook, eds.

[19] J. C. Good, M. Kong, and G. B. Berriman, *OASIS: A Data Fusion System Optimized for Access to Distributed Archives*, Proceedings of Astronomical Data Analysis Software and Systems XII, ASP Conference Series, Vol. 295, 2003, H. E. Payne, R. I. Jedrzejewski, and R. N. Hook, eds.

[20] J. C. Jacob, R. Williams, J. Babu, S. G. Djorgovski, M. J. Graham, D. S. Katz, A. Mahabal, C. D. Miller, R.

Nichol, D. E. Vanden Berk, and H. Walia, *Grist: Grid Data Mining for Astronomy*, Proceedings of Astronomical Data Analysis Software and Systems XIV, ASP Conference Series, Vol. 347, 2005, P. L. Shopbell, M. C. Britton, and R. Ebert, eds.

[21] S. Derriere, F. Ochsenbein, T. Boch, and G. T. Rixon, *Metadata for the VO: The Case of UCDs*, Proceedings of Astronomical Data Analysis Software and Systems XII, ASP Conference Series, Vol. 295, 2003, H. E. Payne, R. I. Jedrzejewski, and R. N. Hook, eds.

[22] The TeraGrid, `http://www.teragrid.org`.

[23] B. Berriman, D. Curkendall, J. C. Good, L. Husman, J. C. Jacob, J. M. Mazzarella, R. Moore, T. A. Prince, and R. E. Williams, *Architecture for Access to Compute Intensive Image Mosaic and Cross-Identification Services in the NVO*, Proceedings of SPIE Astronomical Telescopes and Instrumentation: Virtual Observatories Conference, August 2002.

[24] G. B. Berriman, D. Curkendall, J. Good, J. Jacob, D. S. Katz, T. Prince, and R. Williams, *Montage: An On-Demand Image Mosaic Service for the NVO*, Proceedings of Astronomical Data Analysis Software and Systems XII, ASP Conference Series, Vol. 295, 2003, H. E. Payne, R. I. Jedrzejewski, and R. N. Hook, eds.

[25] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, and M. Livny, *Pegasus: Mapping Scientific Workflows Onto the Grid*, Proceedings of Across Grids Conference, 2004.

[26] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda, *Mapping Abstract Complex Workflows Onto Grid Environments*, Journal of Grid Computing, Vol. 1, No. 1, pp. 25-39, 2003.

[27] J. Frey, T. Tannenbaum, M. Livny, and S. Tuecke, *Condor-G: A Computation Management Agent for Multi-Institutional Grids*, Proceedings of the 10th IEEE International Symposium on High-Performance Distributed Computing, 2001.

[28] E. Bertin, SWarp User's Guide.

[29] D. Makovoz and I. Khan, *Mosaicking with MOPEX*, Proceedings of Astronomical Data Analysis Software and Systems XIV, ASP Conference Series, Vol. 347, 2005, P. L. Shopbell, M. C. Britton, and R. Ebert, eds.

[30] S. G. Djorgovski, R. R. Gal, S. C. Odewahn, R. R. de Carvalho, R. Brunner, G. Longo, and R. Scaramella, *The Palomar Digital Sky Survey (DPOSS)*, in Wide Field Surveys in Cosmology, S. Colombi and Y. Mellier, eds.

[31] M. F. Skrutskie, R. M. Cutri, R. Stiening, M. D. Weinberg, S. Schneider, J. M. Carpenter, C. Beichman, R. Capps, T. Chester, J. Elias, J. Huchra, J. Liebert, C. Lonsdale, D. G. Monet, S. Price, P. Seitzer, T. Jarrett, J. D. Kirkpatrick, J. Gizis, E. Howard, T. Evans, J. Fowler, L. Fullmer, R. Hurt, R. Light, E. L. Kopan, K. A. Marsh, H. L. McCallon, R. Tam, S. Van Dyk, and S. Wheelock, *The Two Micron All Sky Survey (2MASS)*, The Astronomical Journal, Vol. 131, pp. 1163-1183, 2006.

[32] The Flexible Image Transport System (FITS),
`http://fits.gsfc.nasa.gov`, `http://www.cv.nrao.edu/fits`.

[33] E. W. Greisen and M. R. Calabretta, *Representations of World Coordinates in FITS*, Astronomy & Astrophysics, Vol. 395, pp. 1061-1075, 2002.

[34] MPI: A Message Passing Interface Standard, `http://www-unix.mcs.anl.gov/mpi`.

[35] C. A. Beichman, G. Neugebauer, H. J. Habing, P. E. Clegg, T. J. Chester, and The Joint IRAS Science Working Group, *Infrared Astronomical Satellite (IRAS) Explanatory Supplement*, 1988.

[36] J. C. Jacob and L. Plesea, *Fusion, Visualization and Analysis Framework for Large, Distributed Datasets*, IEEE Aerospace Conference, 2001, ISBN 0-7803-6600-X.

[37] J. C. Jacob and L. E. Husman, *Large Scale Visualization of Digital Sky Surveys*, Virtual Observatories of the Future, R. J. Brunner, S. G. Djorgovski, and A. S. Szalay, eds., Astronomical Society of the Pacific Conference Series, Vol. 225, pp. 291-296, 2001.

[38] M. Widenius, MySQL AB, D. Axmark, *MySQL Reference Manual: Documentation from the Source*, O'Reilly, June 2002, ISBN 0-59600-265-3.

[39] MySQL, `http://www.mysql.com`.

[40] The High Performance Storage System (HPSS) at Caltech's Center for Advanced Computing Research,
`http://www.cacr.caltech.edu/resources/HPSS/index.html`.

[41] Hierarchical Storage Interface (HSI) for High Performance Storage Systems (HPSS), `http://www.sdsc.edu/Storage/hsi`.

[42] The Storage Resource Broker (SRB) at San Diego Supercomputer Center (SDSC), `http://www.npaci.edu/SRB`.

[43] J. C. Jacob, G. Block, and D. W. Curkendall, *Architecture for All-Sky Browsing of Astronomical Datasets*, Proceedings of Astronomical Data Analysis Software and Systems XII, Baltimore, MD, Oct. 13-16, 2002, Astronomical Society of the Pacific Conference Series, H. Payne, R. Jedrzejewski, and R. Hook, eds.

Edited by: Marcin Papzycki
Received: April 1, 2003
Accepted: February 21, 2006

Scalable Computing: Practice and Experience
Volume 7, Number 1, pp. 77–93. http://www.scpe.org

ISSN 1895-1767
© 2006 SWPS

A HISTORICAL ANALYSIS OF FIBER BASED OPTICAL BUS PARALLEL COMPUTING MODELS

BRIAN J. D'AURIOL* AND MARIA BELTRAN†

Abstract. A comprehensive overview and survey of the developments in optical bus parallel computing models is presented in this paper. The first model proposed was the APPB in 1990. Since then, in the order of their appearance, the remaining nine models surveyed are: APPBS, ASOS, LPB, RASOB, AROB, LARPBS, PB, LAPOB and PR-MESH. Research trends observed from this analysis indicate periods of model development leading to more and more sophistication and complexity in the model, followed by periods of model simplifications. These periods appears to occur in cycles. We note the widespread and global research interest in these models. The three most popular models appear to be RASOB, AROB and LARPBS. We also have analyzed a crucial aspect of these models, the bus cycle time definitions, and have determined inaccuracies in most of the definitions appearing in the literature. We also provide refinement to the definitions to correct such inaccuracies.

Key words. Optical Bus, Parallel Computing

1. Introduction. Optical fiber bus interconnection parallel computing models were initially proposed over a decade ago. Since then, at least ten distinct models have been developed with many corresponding publications. In addition, related work on implementation as well as routing and addressing have been noted. A principal reason for the success of this research area stems from the advantages of optical communications together with bus-based systems. Some of the advantages include inherent pipelining of messages due to the unidirectional propagation nature of optical signals as well as power speed and crosstalk advantages over electronic buses [1].

A comprehensive overview and survey of the developments in optical bus parallel computing models is presented in this paper. This survey includes ten optical bus models that were proposed between the period of 1990 to 2000. Many publications have appeared and publications continue to be submitted based on these models. First, a description of the salient parts of optical buses is given. Next, the ten surveyed models are described. Observations regarding similarities and differences inherent in the models are noted. Lastly, some analytical comments are presented.

Several papers provide survey-type information. These survey papers [2, 1] are not reported in the model summary sections. This paper actually extends the work of the above survey papers.

The purposes of this paper are three-fold. First, to provide to the computer professionals who do not have detailed knowledge of optical bus parallel models an introduction and overview of the major points of these models as well as providing information about the various proposed models. Second, to provide to the research community a handy-reference of first citations, categorizations of existing publications and a source of additional material to consider. Third, to provide to the research community an extensive literature bibliography.

The paper is organized as follows. A review of basic concepts in optical bus parallel models is given in Section 2. In Section 3, a brief overview of ten optical bus models with an emphasis on bus cycle is given. Historical comments are made in Section 4. An analysis of bus cycles definitions in these models is given in Section 5. Conclusions are given in Section 6

2. Optical Bus Model. In general, an optical bus model uses one or more optical waveguides to connect a linear array of N processors labeled 0 through $N-1$. This architecture can be extended to more than one dimension where, for example, processors are arranged in a matrix. Processors are connected to the waveguides by injectors, which inject light pulses onto the waveguide(s), and detectors, which detect light pulses on the waveguide(s). The time it takes a pulse to traverse the distance between any two adjacent injection points (or two detection points) is a constant commonly referred to as τ, while the duration of the pulse is usually referred to as ω. In the literature, one simple case of collision is addressed by defining b as the maximum size of a message such that $b\omega < \tau$.

There are four aspects of optical bus models that may be used as criteria for classification: (1) the number of buses to which processors are connected, (2) the type of bus that is used (folded or non-folded), (3) the dimension of the model, and (4) the type of addressing used. These aspects are detailed below.

In order to enable all processors arranged in a linear array to communicate with each other, the interconnection network must allow traffic to travel in two directions. Due to the unidirectional propagation property

* (dauriol@acm.org).
†CAS Inc., El Paso, TX, USA (maria.beltran@cas-west.com).

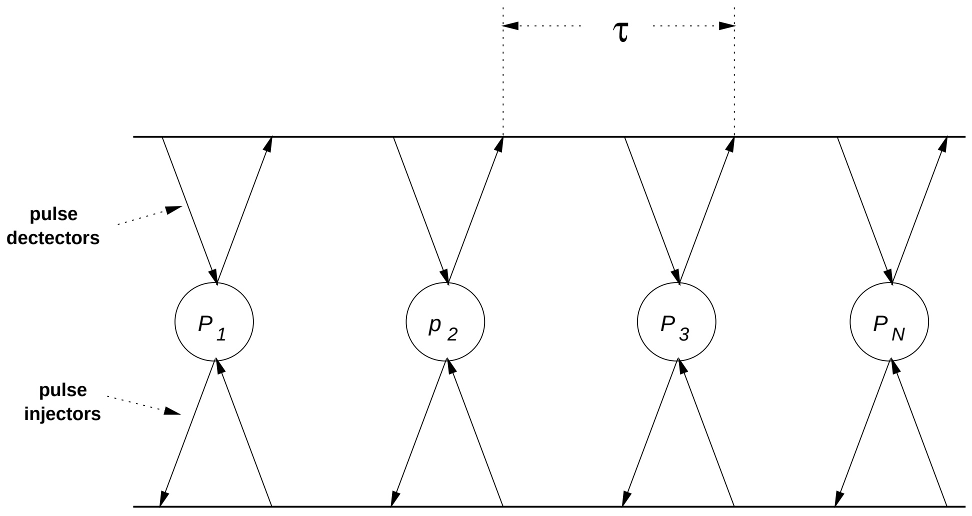


Fig. 2.1. *Non-folded (Two) Bus Configuration*

of light, however, a single optical bus running along the length of the array will only allow communication in one direction. The directional requirement is fulfilled by using two buses. Figure 2.1 illustrates this. Note that processors are connected to both buses by both injectors and detectors. The two buses in this configuration are referred to as non-folded buses. The folded bus, on the other hand, combines both directional requirements into a single bus. It is a single bus that parallels the array of processors, and is folded around one of the ends of the array. This enables light to travel in both directions with respect to the processors. Typically, the linear segment before the fold is called the transmitting segment while that after the fold is called the receiving segment. On the transmitting segment, processors are connected by injectors and on the receiving segment, by detectors. The time it takes a pulse to traverse the fold of the bus is a constant that is denoted as γ, and is not necessarily a multiple of either τ or ω. Figure 2.2 shows the folded bus architecture.

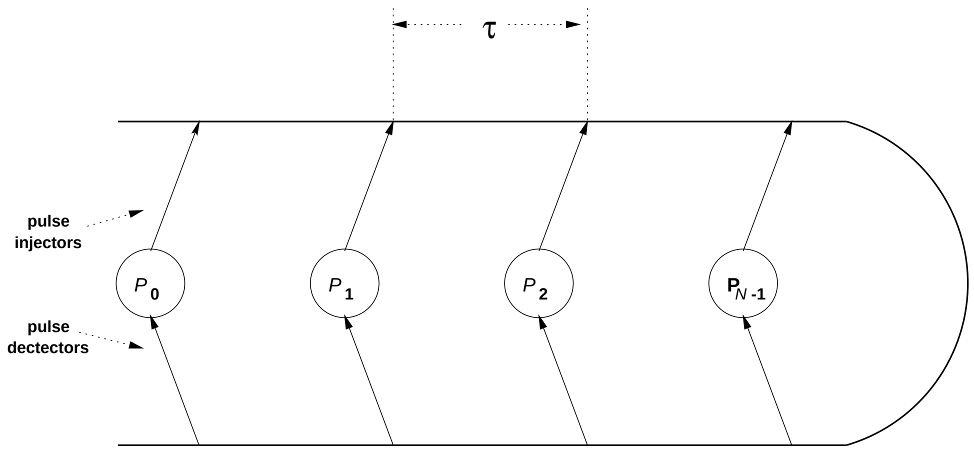


Fig. 2.2. *Folded (Single) Bus Configuration*

This description of the optical bus architecture applies to one-dimensional (or 1-D) models. This architecture is used as a building block for models consisting of more than one dimension. The most common multi-dimensional model found in the literature is the two-dimensional (or 2-D) model. In such a model, processors are arranged in a matrix format and the optical buses belong to one of two groups, rows or columns. The orientation of the row buses is perpendicular to that of the column buses. The two-dimensional model has been found to be helpful in reducing time delays in message delivery. Depending on the model, there are various ways that row and column buses can be connected to each other and to the processors. Reconfigurable switches are commonly used to make connections at intersection points. Examples of the use of reconfigurable switches can be found in [3, 4, 5, 6, 7].

The literature search indicates that, in models that use more than one waveguide, models with three waveguides are the most commonly used configuration. Coincident pulse addressing, discussed subsequently, is the primary reason why three waveguides are used. Most of the models that employ more than one waveguide provide one for the message and the remaining two for addressing purposes. The two addressing waveguides it uses are referred to as the select and reference waveguides.

Coincident pulse (CP) is the most common form of addressing. Delays of duration ω are placed between every pair of detectors on the reference and message waveguides to implement CP. Addressing works as follows: first, the source processor sends a pulse on the reference waveguide at the same time it starts to send the message on the message waveguide. Then the processor waits a factor of ω, depending on the destination processor it wishes to communicate with, to send a pulse on the select waveguide. The pulse on the reference waveguide is delayed by the delays on the receiving segment such that it will be detected by the intended destination processor at the same time as the select pulse. At that point the message, which also arrives at the intended processor at the same time as the select and reference pulses, is read in from the message waveguide. Figure 2.3 shows a folded bus with the coincident pulse addressing components.

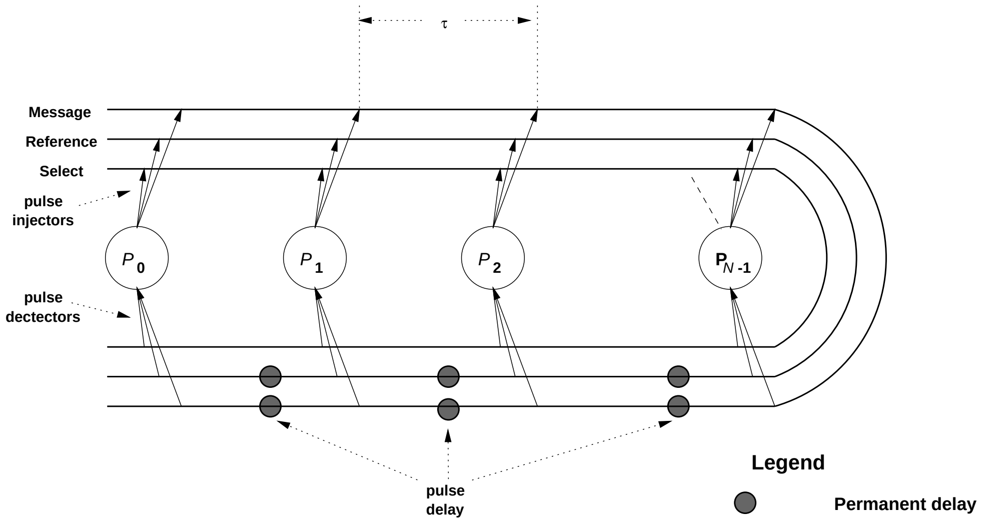


Fig. 2.3. *Folded Bus with Coincident Pulse*

The other commonly used addressing method is time-division multiplexing (TDM). This method involves assigning a particular time slot for a particular processor. This time slot can either be used to send (time-division source multiplexing, or TDSM) or receive (time-division destination multiplexing, or TDDM) optical signals. More detail regarding these bus aspects can be found in [2, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17].

The next section briefly describes the various bus models found in the literature. A point of interest in each section is the information regarding the use of the term ‘bus cycle’. A subsequent section critiques the overall use of this term.

3. Optical Buses in the Literature. This section surveys ten optical bus models found in the literature. The first one was proposed in 1990 while the last in 1998. Figure 3.1 depicts the optical bus model development in a timeline.

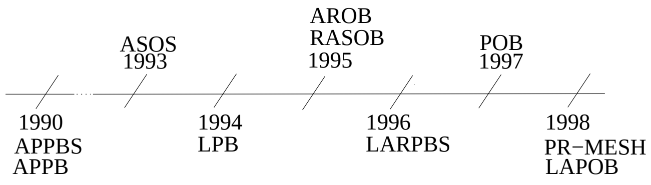


Fig. 3.1. *Optical Bus Model Timeline*

3.1. APPB (1990). The first proposed optical bus model is the Array of Processors with Pipelined Buses (APPB) [3], which can be one or two dimensions. In one dimension, there are two optical buses that are placed parallel to and on either side of a linear array of N processors. Each processor is connected to each bus via one injector and one detector. This allows a processor to communicate with any other processor. The bus cycle time is $N\tau$ time units. In two dimensions, each processor is connected to four buses by a switch, two row buses and two column buses. For this configuration, a bus cycle is defined as $N_1\tau$ for a row bus and $N_2\tau$ for a column bus. In addition, the authors define a *petit bus cycle* as τ. APPB uses either TDM or CP for addressing and routing. For the TDM method in the one dimensional configuration, a set of two control functions, *send* and *wait*, are used to control access to the optical bus. The two dimensional APPB adapts the *send* and *wait* and introduces the *relay* function to provide control of messages between different rows (messages travel row-wise first). Figure 3.2 illustrates the one dimension architecture, while Figure 3.3 illustrates the two-dimensional architecture. Individual waveguides are not shown. Rather, the optical bus is represented by a single line. Variations of APPB that incorporate folded bus and conditional delay switches are discussed in [18]. Papers reporting work on APPB are [3, 19, 20, 21, 22, 23, 24, 25, 26, 27].

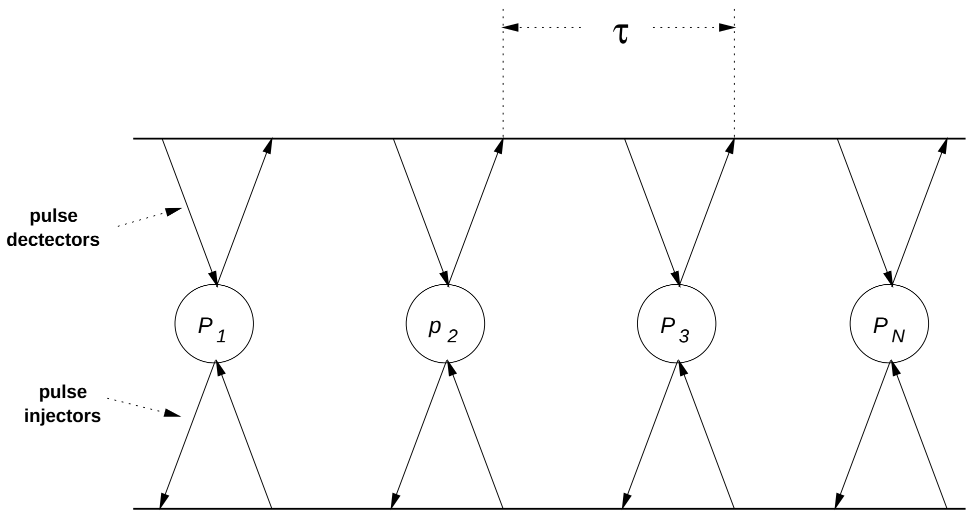


Fig. 3.2. *APPB 1-D Architecture*

3.2. APPBS (1990). The Array of Processors with Pipelined Buses Using Switches (APPBS) [3] is the same as the 2-D APPB; with one important difference. In this model, switches are used at every intersection of row and column buses, thus, eliminating the need for processors to act as relays between the buses. As with the APPB, APPBS can use TDM or CP addressing methods and the bus cycle is $(N_1 + N_2)\tau$. Switch configurations impact the communication. Each of the switches can be configured as straight or crossed and can be dynamically reconfigured relative to the start of the bus cycle (by using petit cycles). This flexibility requires additional conditions to ensure collision-free communication, especially when messages need to be switched at the same place and time. Algorithms such as matrix transpose, binary tree routing, and perfect shuffle are implemented in a one step operation. This one step not only includes bus cycle time but also the time to process some messages and the switch setup time. Figure 3.4 illustrates this model. Individual waveguides are not shown. Papers reporting on APPBS are [3, 20, 23, 24, 28, 29].

3.3. ASOS (1993). The Array Structure with Synchronous Optical Switches (ASOS) [4] is a two-dimensional model that uses multiple folded buses. This type of architecture consists of a single waveguide folded around a linear array of N processors. A 2×2 switch is placed at the intersections of receiving segments of row buses and transmitting segments of column buses. Each processor is connected to the transmitting and receiving segments of the row bus, but only connected to the receiving segment of the column bus. ASOS uses a combination of TDSM and TDDM to route messages between rows and columns; and the CP method for destination addressing. The term bus cycle is not explicitly mentioned, however, the end-to-end propagation delay of a row bus is mentioned and is defined as $(2N-1)D$ (where D roughly corresponds with the τ elsewhere). This equation includes the delay for the fold, which is set to D. One additional architectural feature is due

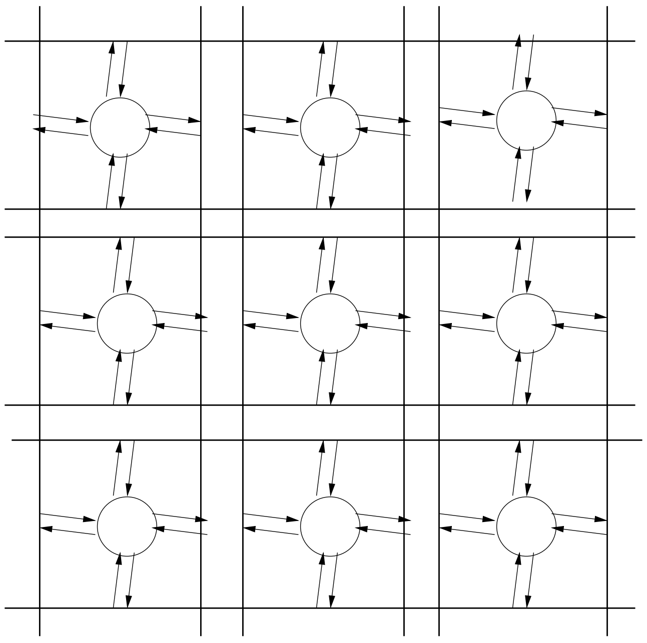


Fig. 3.3. *APPB 2-D Architecture*

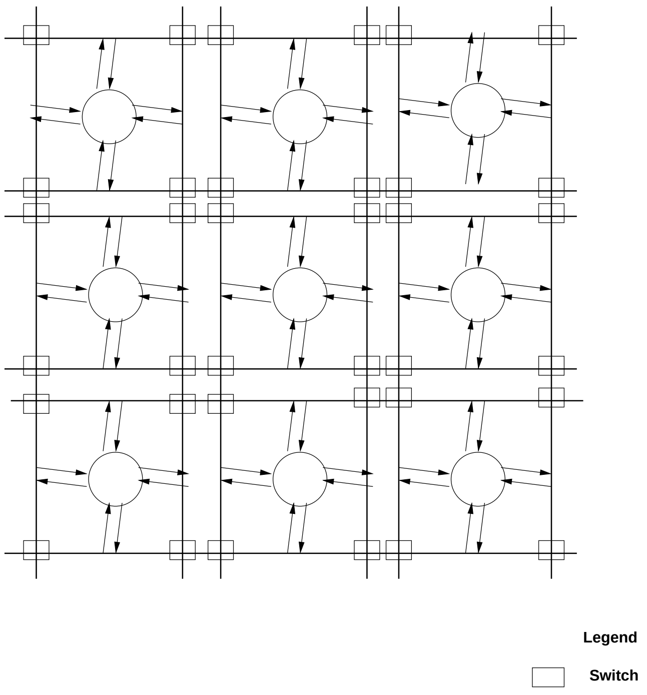


Fig. 3.4. *APPBS Architecture*

to the bus contention that occurs if more than one processor sends a message to the same column bus. To alleviate this, a *reservation* waveguide is included in each row bus. Processors needing to send messages use this waveguide to determine if there already is a message that would contend with theirs. Priority schemes are used to establish conflict resolution. Figure 3.5 illustrates this architecture. Individual waveguides are not shown. A modified architecture called the Symmetric ASOS (SASOS) is presented in [30]. We point out that the authors in [6] refer to this model as ASOB; more likely, the authors appear to refer to the RASOB model (see below).

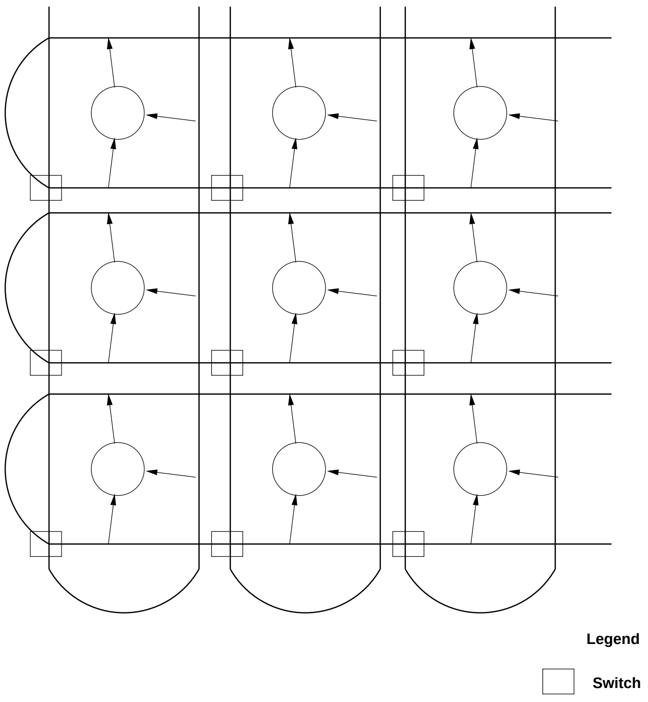


Fig. 3.5. *ASOS Architecture*

3.4. LPB (1994). The Linear Pipelined Bus (LPB) model appears in 1994 in [8]. There is some contradicting evidence in the literature pertaining to the definition and origin of the LPB model. The authors of [28] cite [31], which has similar title, as a source for this optical bus model. However, upon a review of that paper, it is concluded that the content does not include optical buses. It is possible that the authors inadvertently cited the 1995 paper instead of the 1994 paper. The author of [32] cites reference #59, a submitted paper as the source for LPB, yet that paper was not published as cited. It is possible that, in reality, that submitted paper is [33] (which has a different title but the same authors, albeit, in a different order). Personal communication with the author of [32] confirms that the submitted paper was indeed published but under a different title and different author order. In [33], the authors cite the 1994 paper as the source of LPB. Hence, it is concluded that the 1994 publication [8] is the original source of LPB.

LPB is a one-dimensional model that uses the folded bus. It not only has the fixed delays on the receiving segments of the reference and message waveguides, but also has conditional delays between every pair of injection points on the select waveguide. A bus cycle is defined as the end-to-end propagation delay on the bus and is presented prior to including the use of conditional delays (which make the end-to-end propagation delay a variable). The bus cycle formula is defined as $2N\tau + (N-1)\omega$. Figure 3.6 illustrates this architecture. Individual waveguides are shown, including the placement of delays. $S_1, \ldots, S_{N-1}$ denote the conditional delays, each controlled by processors $P_1, \ldots, P_{N-1}$, respectively. We note the similarity with the LARPBS model described later; in comparison, this model is not reconfigurable and uses a slightly different addressing technique. Papers reporting on LPB are [8, 34, 33, 31].

3.5. RASOB (1995). The Reconfigurable Array with Spanning (or Slotted) Optical Buses (RASOB) [5] is similar to ASOS. The architecture was initially designed to support SIMD processing and to contain less complexities than ASOS. It uses the folded bus, and can be one or two dimensional. In the former, each processor is connected to the bus on the transmitting and receiving sides. In the latter, there is an $N \times N$ matrix of folded buses. As in ASOS, one 2×2 switch is placed at each intersection of row and column buses. Each processor is connected to the buses: two connections for receiving (one for row, one for column) and one for transmitting. A constraint that no more than one processor per row can send a message to processors in the same column exists in a communication cycle. Either a TDDM or TDSM method is used for addressing. The authors state that support for MIMD processing could be accomplished by using coincident pulses, however, doing so would make the architecture more complex. The bus cycle for RASOB is defined as $4N\tau$ where a row

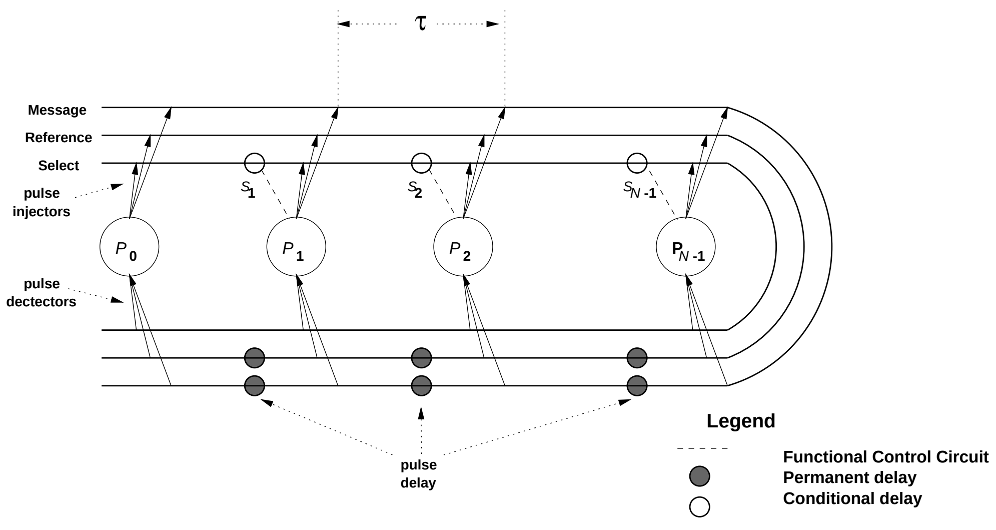


Fig. 3.6. *LPB Architecture*

bus has $2N\tau$ cycle time. Figure 3.7 illustrates this architecture. The figure shows a 3×3 RASOB. Individual waveguides are not shown. Papers that report on RASOB are [5, 35, 36, 37, 38, 39, 40, 41, 42, 43].

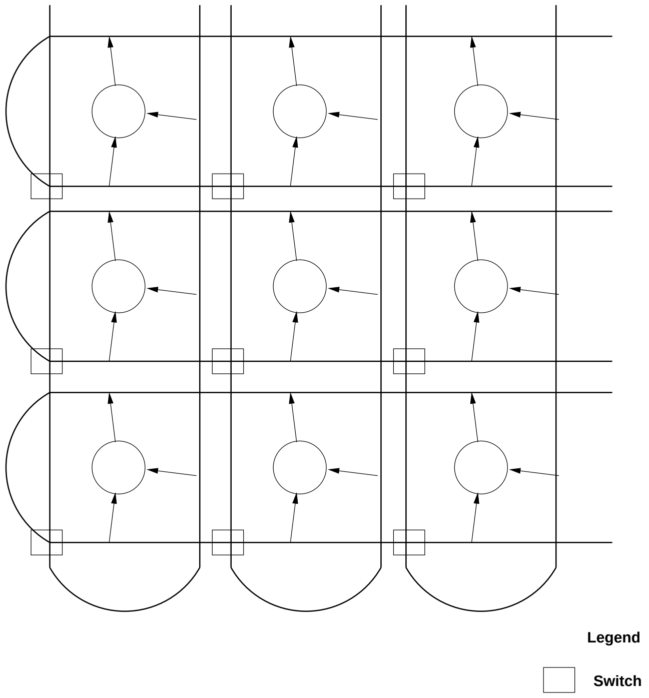


Fig. 3.7. *RASOB Architecture*

3.6. AROB (1995). The Array with Reconfigurable Optical Buses (AROB) [6] also uses the folded bus. The AROB is an $N_1 \times N_2$ reconfigurable mesh where each processor contains registers for the temporary storage of messages being routed. Processors use an internal switching system to reconfigure the network by connecting or disconnecting four I/O ports to each other. Two of the ports are connected to either segment of the row bus while the other two ports are connected to the column bus. Both TDM and CP can be used for addressing. Notable is that communication involves *counting* by processors. Thus, the communication time must also incorporate some processing time overhead. Since the processing time can be orders of magnitude greater than the end-to-end propagation time, the processing time order must be of the same magnitude as that of the end-to-end propagation time: this is achieved by the condition that the number of processors must be greater than the ratio of the processing time to communication time (subject to an appropriate bus length). Two other features of the AROB are its bit polling capability and its capability to introduce/adjust signals by multiple unit delays per processor [28, 33]. Bit polling is the ability to select the k^{th} bit of a group of messages and determine the number of one bits. Figure 3.8 illustrates this architecture. The figure shows a 3×3 AROB. Individual waveguides are not shown. Papers that report work on AROB are [6, 25, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 29, 28, 61, 62].

A $1 \times N$ (one-dimensional) AROB is commonly referred to as a Linear AROB or LAROB. The bus cycle time for LAROB is defined slightly differently in different papers. In [6] it is defined to be the end-to-end propagation delay: $2N\tau$ plus some processing time. In [57], it is defined slightly differently, to be only the end-to-end propagation delay: $2N\tau$. Each processor controls its pair of bus connection switches. When these switches are set crossed at a processor, the bus is split into two at that point. In [60], AROB is extended to multi-dimensions.

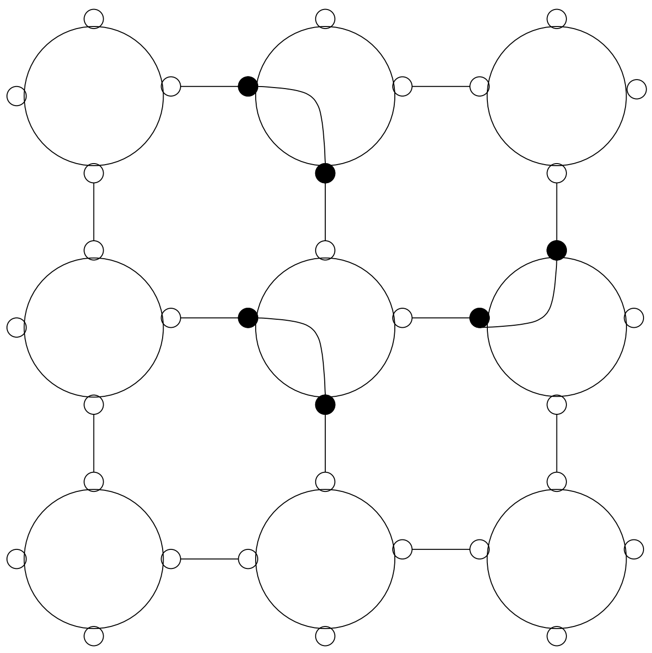


Fig. 3.8. *AROB Architecture*

3.7. LARPBS (1996). The Linear Array with a Reconfigurable Pipelined Bus System (LARPBS) [63] is a one dimensional reconfigurable folded bus model. It contains $N-1$ fixed delays on the receiving side of the reference and message waveguides and $N-1$ conditional delays on the transmitting side of the select waveguide. Switches allow partitioning of the bus. Addressing can be done by either TDM or CP. The bus cycle is the end-to-end propagation delay on the bus: $2N\tau + (N-1)\omega$. Unlike AROB, LARPBS does not allow counting by processors. However, at the beginning of a bus cycle, each processor must set the conditional switches. Due to this (and other factors), the authors claim that LARPBS, unlike theoretical models such as PRAM, can be practically implemented by current (as of 1996) optical technology. Refer to [64] for a current discussion on implementation technology. Figure 3.9 illustrates this architecture. Individual waveguides are shown, including the placement of delays. $S_1 \ldots S_{N-1}$ denote the conditional delays, each controlled by processors $P_1, \ldots, P_{N-1}$, respectively. B_i^t and B_i^r, $0 \leq i \leq N-2$, denote the pair of switches controlled by P_i which partition the bus. Papers reporting on LARPBS are [63, 65, 9, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 32, 78, 79, 80, 81, 82, 34, 83, 33, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 64, 98, 99, 100, 101, 102, 103, 104].

3.8. POB (1997). The Pipelined Optical Bus (POB) [10] model is a one-dimensional folded bus model. It contains N conditional delays on the receiving side of the reference waveguide that are controlled by processors

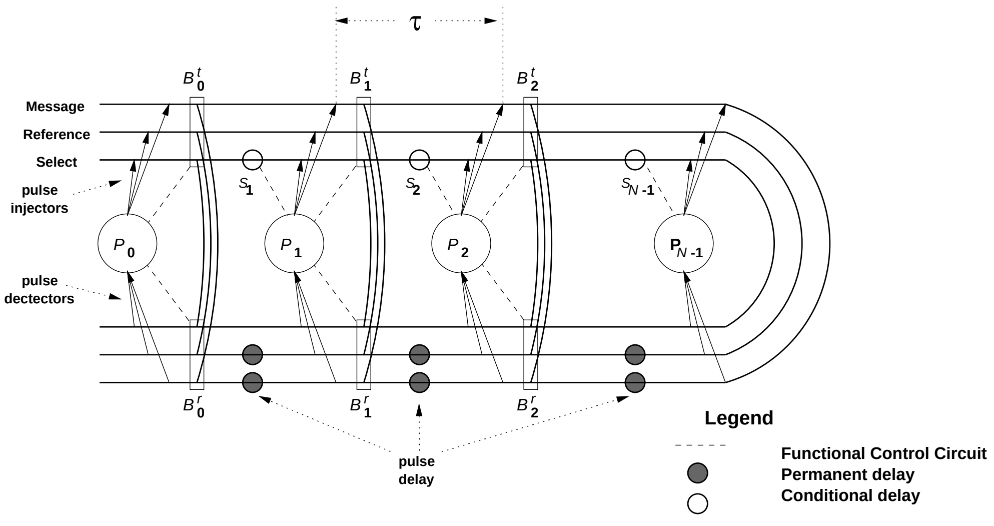


FIG. 3.9. *LARPBS Architecture*

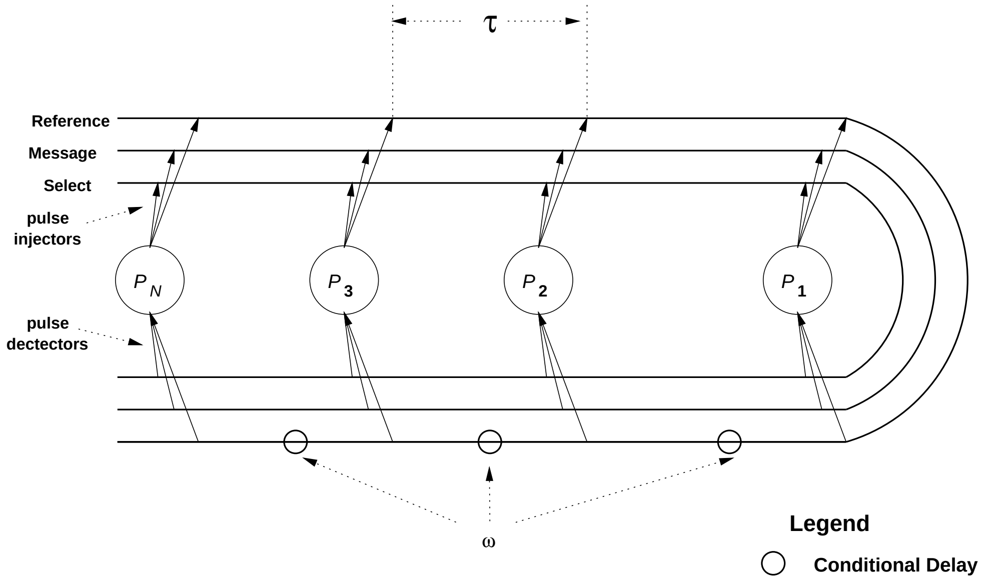


FIG. 3.10. *POB Architecture*

as needed. Addressing is done with either TDSM or CP. There is a possible problem that may occur when a message and a coincident pulse arrive at a destination processor at the same time: during the detection time interval for a coincidence pulse, part of the message could have already passed by. The *offset message transmission scheme* (OMTS) addresses this problem by sending the message a little after the select pulse (the end of the previous message stream is allowed to overlap with the next set of addressing pulses). The bus cycle time is the time that it would take N consecutive packet slots to propagate along the bus. The length of a packet slot is measured in time units, i. e., the larger of the length of the message or the total length of its associated sequence of select pulses: at most $D = \tau$ units. The model is described by the authors as "more powerful" than APPB and "more cost-effective" than LARPBS. Figure 3.10 illustrates this architecture. Individual waveguides are shown, including the placement of delays. Papers that report on POB are [10, 105, 15, 34, 33]

3.9. LAPOB (1998). The Linear Array of Pipelined Optical Buses (LAPOB) [11] is a one-dimensional folded bus model. It uses the CP addressing technique. Besides the fixed delays on the receiving segment, there are no other delays or switches on the bus. Each processor can only send one message in a single bus cycle. However, each processor contains special electronic hardware that allows it to address multiple processors if the

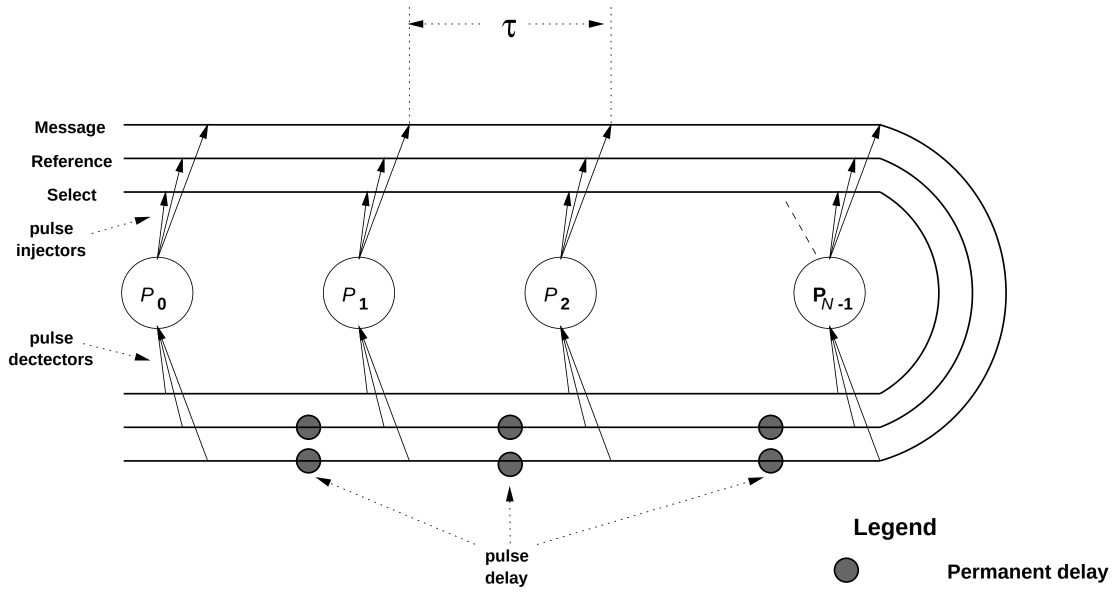


FIG. 3.11. *LAPOB Architecture*

positions of the destination processors form one of two patterns: contiguous interval (a sequence of adjacent processors) or regularly spaced. The formula for a bus cycle is not given in the paper. One advantage is that reconfiguration hardware is unneeded and, because of this, the model is less complex. Figure 3.11 illustrates this architecture. Individual waveguides are shown, including the placement of delays.

3.10. PR-MESH (1998). The Pipelined Reconfigurable Mesh (PR-Mesh) [7] uses the LARPBS as a building block to make up a k-dimensional mesh. The one dimensional PR-MESH is identical to LARPBS. For the k dimension, each processor has $2k$ ports connected to buses. It uses the CP addressing technique. The bus cycle is described as the end-to-end propagation delay as presented in APPB and LARPBS. In the two dimensional mesh, each processor controls four sets of switches, one set for each intersection of the buses. These switches can be configured in one of ten different ways. This allows traffic to flow differently between a total of four different transmitting segments and four different receiving segments. Figure 3.12 illustrates this architecture. Individual waveguides are shown, including the placement of delays. Papers that report on PR-MESH are [7, 106, 28, 29].

4. Historical Analysis. Some analysis was conducted on the models found in the literature. It includes observations about comparisons between the models, the types of algorithms proposed, the popularity of the models, trends in architecture complexity of the models and some miscellaneous observations of interest.

Theoretical comparisons between several of the optical bus models as well as with respect to PRAM have been published. These comparisons seek to establish both the functional equivalence and the relative strengths of models. Several comparisons are reported in [6]: AROB is compared with reconfigurable networks; and, APPB is compared with PRAM. LARPBS is compared with PRAM in [83]. The authors in [7] compare the PR-MESH model with other reconfigurable models on the basis of complexity classes. One interesting result is "...the contribution of pipelining to the [PR-MESH] model is limited to no more than duplicating buses in the [Linear Reconfigurable Network]..." The equivalence of LPB, POB, and LARPBS is reported in [33]. And, the algorithm complexities of the PR-Mesh, APPBS, and AROB are determined to be same as for the LR-Mesh and CF-LR-Mesh [29]. Additional comparison of the PR-MESH with the linear reconfigurable mesh appears in [107]. Some limited architectural comparisons are also made in [32].

Many of the papers surveyed describe algorithms for the respective models. In several cases, see for example [6, 63, 32], communication and computation algorithms are developed as primitives to be used in more sophisticated algorithms. These include binary prefix sums and compaction. In [71], some of these primitives for the LARPBS model are formalized in a lemma. In general, algorithms that have been developed include sorting and selection, matrix calculations (including the Four Russians' algorithm for boolean matrix multiplication), neural networks, and image processing (including vector median filter, Hough transform and the Euclidean Dis-

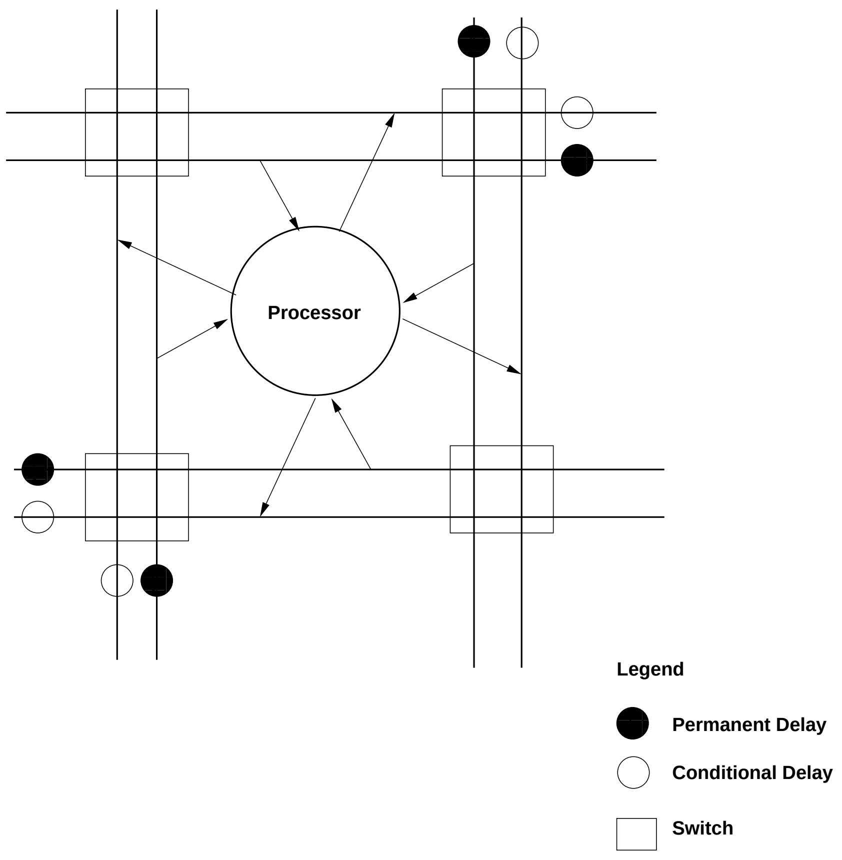


FIG. 3.12. *PR-MESH Architecture (1 Node)*

tance Transform). Many algorithms exhibit static communication decomposition, that is, the communication pattern is statically determined during algorithm development. This is consistent with both the allocation of communication in terms of bus cycles as well as algorithm development based on primitive operations.

Trends were observed in the popularity of models according to their level of complexity. The analysis included categorizing the number of publications reported per model and per year. Refer to Table 4.1 for the complete list of models and number of publications. In judging the complexity, or sophistication, of a model, the following features are considered: multiple dimensionality, the use of a folded bus, the use of switches, the use of coincident pulse addressing, and bus partitioning. The greater number of these features that a model contains, the more complex and sophisticated it is considered. This informal criterion is used to guide the trend analysis.

From 1990 to 1993, an increase in the sophistication and complexity of the early models is observed. This was followed by a period of simplification (1993-1995). A jump in the sophistication is noted in 1995 with the AROB model. Again, a period of simplification follows (1996-1998) with another jump noted for the PR-MESH model. It is noted that this analysis is consistent with the intention of RASOB as stated in Section 3.5, that is, RASOB was designed to have fewer complexities than ASOS.

The popularity of these models is observable from the number of publications listed in Table 4.1 (some publications are common to two or more models). The three most popular are: RASOB, AROB and LARPBS.

Combining these results, it is suggested that researchers are more strongly attracted to something between the simple and the complicated. It is also suggested that the capabilities incorporated into the models during the middle period, 1995-1996, are sufficient for supporting current research interests. It is worthy to point out that the recent PR-MESH model may indeed signify a future research movement to consider models of higher degrees of sophistication and complexity. If so, then this may also indicate that current research has explored many of the issues of the optical bus parallel model and is ready to consider additional challenges. However, these latter comments must be interpreted as highly speculative given the lack of data to support such a trend.

Table 4.1
Model Publications

Model	**Number of Publications**	**Year of First Publication**
APPB	10	1990
APPBS	6	1990
ASOS	1	1993
LPB	4	1994
RASOB	10	1995
AROB	23	1995
LARPBS	46	1996
POB	5	1997
LAPOB	1	1998
PR-MESH	4	1998

Perhaps in several years, additional publication history could support or refute such speculation.

During the course of this analysis, several other interesting observations were noted. One publication reporting work on ASOS was located, yet, it is cited in many publications. Some papers claim that to provide for MIMD algorithms, additional complexities would need to be incorporated. Other optical buses can be found in the literature, for example, free space optical buses as well as NASA's ROBUS as part of the SPIDER architecture. These models do not appear to follow the architectural approach of the models surveyed in this paper and therefore were not included into this paper.

Some recent developments are: a) the restricted LARPBS (RLARPBS) model [99] proposed to more accurately model time analysis of algorithms, b) the parameterized LARPBS (LARPBS(p)) model [103] proposed as a bridging model and c) a generic optical bus model [108] proposed to capture some of the common features of the models surveyed in this paper for MIMD communication analysis.

5. Bus Cycle Issues.

In the course of the analysis, bus cycles have been noted to play a crucial role in the algorithmic evaluations on these models. In many cases, algorithm complexity is described as order Omega of bus cycle, for example, constant time bus cycle complexity for the binary prefix sums algorithm on the LARPBS. Closer examination of bus cycle definitions on many of the models reveal inconsistencies between the architecture and the definition, that is, the definition appears to represent a simplified architecture. Investigating this further, it is noted that nearly all of the algorithmic work referencing bus cycle is given in the context of asymptotic analysis expressions, wherein only the dominate term needs to be expressed. Although not in all cases, it is noted that the definitions as given in the literature are consistent with such a use.

A refinement of the bus cycle definitions for all the models is presented in this section. A general expression template is formulated which is then applied to each architecture. In general, the folded bus cycle equation is composed of four parts. Part 1 of the equation corresponds to the length of the transmitting and receiving portions of the folded bus. Part 2 corresponds to the curved portion of the bus, while Part 3 corresponds to the delays on the bus. Part 4 corresponds to the portion of the bus that remains past the detection point of the last processor on the receiving side of the bus. Recall from Section 2 that $b\omega < \tau$. To efficiently use the bus bandwidth, $b\omega$ should be maximized. To model this requirement, the factor $\tau - \epsilon$ is introduced, and may be approximated as simply τ. Table 5.1 presents the refined bus cycle definitions as compared with those given in the literature for each of the models. In the Refined Bus Cycle column, 's', 'r' and 'm' refer to the select, reference and message waveguides, respectively.

Note that, in cases where the model is only 2-D, one of the 1-D 'pieces' is used to compute the bus cycle; in cases where the model is 1-D or multidimensional, the 1-D variety is used. On the PR-MESH, this is for $k = 1$ (the 1-D case). S_i denotes each of the conditional delays, where $S_i = 0$ means the delay controlled by P_i is turned off and $S_i = 1$ means the delay is turned on. The processor time component is denoted by ϕ. For ASOS, we interpret the D parameter in terms of the corresponding time parameter τ.

TABLE 5.1
Bus Cycle Equations

Model	**Literature Bus Cycle**	**Refined Bus Cycle**
APPB	$N\tau$	$(N+1)\tau$
APPBS	$N\tau$	$(N+1)\tau$
ASOS	$2(N-1)\tau$	$2(N-1)\tau+\gamma+\tau+\phi$
RASOB	$2N\tau$	$2(N-1)\tau+\gamma+\tau$
LPB	$2N\tau+(N-1)\omega$	s: $2(N-1)\tau+\gamma+\omega\sum S_i+\tau$ r,m: $2(N-1)\tau+\gamma+(N-1)\omega+\tau$
AROB	$2N\tau+\phi$	s: $2(N-1)\tau+\gamma+\tau$ r,m: $2(N-1)\tau+\gamma+(N-1)\omega+\tau+\phi$
LARPBS	$2N\tau+(N-1)\omega$	s: $2(N-1)\tau+\gamma+\omega\sum S_i+\tau$ r,m: $2(N-1)\tau+\gamma+(N-1)\omega+\tau$
POB	N/A	s,m: $2(N-1)\tau+\gamma+\tau$ r: $2(N-1)\tau+\gamma+\omega\sum S_i+\tau$
LAPOB	N/A	s: $2(N-1)\tau+\gamma+\tau$ r,m: $2(N-1)\tau+\gamma+(N-1)\omega+\tau$
PR-MESH	see APPB & LARPBS	s: $2(N-1)\tau+\gamma+\omega\sum S_i+\tau$ r,m: $2(N-1)\tau+\gamma+(N-1)\omega+\tau$

6. Conclusions.

This paper provides a comprehensive overview and survey of the developments in optical bus parallel computing models. The first model proposed was in 1990 and since then, ten distinct parallel computing models have been proposed.

Specifically, the research trends we have observed indicate periods of model development leading to more and more sophistication and complexity in the model, followed by periods of model simplifications. These periods occur in cycles. With the many publications surveyed, we note the widespread and global research interest in these models. The three most popular models appears to be RASOB, AROB and LARPBS. We also have analyzed a crucial aspect of these models, the bus cycle time definitions. We have determined inaccuracies in most of the definitions appearing in the literature, although, for most of the applications such definitions have been used for, these inaccuracies appear not to be significant. In the interests of clarifying the correct definitions as well as to support our current research work, we have also provided refinements to the bus cycle definitions.

7. Acknowledgements. Much of this work was conducted while the authors were at The University of Texas at El Paso. We thank the Reference Library staff at Kent State University for their assistance in locating citations.

REFERENCES

[1] S. Sahni, “Models and algorithms for optical and optoelectronic parallel computers,” *International Journal of Foundations in Computer Science*, vol. 12, pp. 249–264, June 2001.

[2] S. Sahni, “Models and algorithms for optical and optoelectronic parallel computer,” in *proc. of 1999 International Symposium on Parallel Architecture, Algorithms and Networks (I-SPAN'99).*, pp. 2–7, June 1999.

[3] Z. Guo, R. G. Melhem, R. W. Hall, D. M. Chiarulli, and S. P. Levitan, “Array processors with pipelined optical busses,” in *Proc. 3rd Symposium on Frontiers of Massively Parallel Computation (Cat. No.90CH2908-2)* (J. Jaja, ed.), (College Park, MD, USA), pp. 333–342, October 1990.

[4] C. Qiao and R. G. Melhem, “Time-division optical communications in multiprocessor arrays,” *IEEE Transactions on Computers*, vol. 42, pp. 577–590, May 1993.

[5] C. Qiao, “Efficient matrix operations in a reconfigurable array with spanning optical buses,” in *Proceedings. Frontiers '95. The Fifth Symposium on the Frontiers of Massively Parallel Computation (Cat. No.95TH8024). IEEE Comput. Soc. Press. 1994*, (McLean, VA, USA), pp. 273–280, Feb 1995.

[6] S. Pavel and S. G. Akl, “On the power of arrays with reconfigurable optical buses,” *Technical Report No. 95-374, Queens University, Kingston, Ontario, CANADA*, February 1995.

[7] J. L. Trahan, A. G. Bourgeois, and R. Vaidyanathan, “Tighter and broader complexity results for reconfigurable models,” *Parallel Processing Letters*, vol. 8, no. 3, pp. 271–282, 1998.

[8] Y. Pan, "Order statistics on optically interconnected multiprocessor systems," *Optics and Laser Technology*, vol. 26, pp. 281–287, August 1994.

[9] Y. Pan and M. Hamdi, "Quicksort on a linear array with a reconfigurable pipelined bus system," in *Proc. of the IEEE Intrnational Symposium on Parallel Architectures, Algorithms, and Networks*, pp. 313–319, June 1996.

[10] Y. Li, Y. Pan, and S. Zheng, "A pipelined TDM optical bus with conditional delays," in *Proceedings of the Fourth International Conference on Massively Parallel Processing Using Optical Interconnections* (J. Goodman, S. Hinton, T. Pinkston, and E. Schenfeld, eds.), (Montreal, Canada), pp. 196–201, June 1997.

[11] H. ElGindy, "An improved sorting algorithm for linear arrays with optical buses (extended abstract)," *(Manuscript)*, April 1998.

[12] D. M. Chiarulli, R. G. Melhem, and S. P. Levitan, "Using coincident optical pulses for parallel memory addressing," *IEEE Computer*, vol. 20, pp. 48–58, Dec. 1987.

[13] S. Zheng, K. Li, Y. Pan, and M. C. Pinotti, "Generalized coincident pulse technique and new addressing schemes for time-division multiplexing optical buses," *Journal of Parallel and Distributed Computing*, vol. 61, pp. 1033–1051, August 2001.

[14] C. Qiao, R. G. Melhem, D. M. Chiarulli, and S. F. Levitan, "Optical multicasting in linear arrays," *International Journal of Optical Computing*, vol. 2, pp. 31–48, April 1991.

[15] S. Zheng and Y. Li, "Pipelined asynchronous time-division multiplexing optical bus," *Optical Engineering*, vol. 36, pp. 3392–3400, December 1997.

[16] S. Zheng, , and Y. Li, "A pipelined TDM optical bus with improved performance," in *Proceedings. Third International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN '97)* (F. Lai, B. Maggs, and D. Hsu, eds.), (Taipei, Taiwan), pp. 49–55, December 1997.

[17] S. Q. Zheng, K. Li, Y. Pan, and M. C. Pinotti, "New addressing schemes for pipelined optical buses," in *6th International Conference on Parallel Interconnects (PI'9)* (M. Haney, R. Kostuk, C. Lund, and E. Schenfield, eds.), (Anchorage, AK, USA), pp. 230–237, October 1999.

[18] Z. Guo, "Sorting on array processors with pipelined buses," in *Proceedings of the 1992 International Conference on Parallel Processing*, pp. 280–292, 1992. Vol. 3.

[19] Z. Guo, R. Melhem, R. Hall, D. Chiarulli, and S. Levitan, "Pipelined communications on optical busses," in *Proceedings of Spie—the International Society for Optical Engineering*, (Boston, MA, USA), pp. 415–426, Nov 1990.

[20] Z. Guo and R. G. Melhem, "Embedding pyramids in array processors with pipelined busses," in *Proceedings of the International Conference on Application Specific Array Processors (Cat. No.90CH2920-7)* (S.-Y. Kung, E. Swartzlander, J. Fortes, and K. Przytula, eds.), (Princeton, NJ, USA), pp. 665–676, September 1990.

[21] G. Zicheng, R. Melhem, R. Hall, D. Chiarulli, and S. Levitan, "Pipelined communications on optical busses," in *Proc. of Spie*, pp. 415–26, 1991.

[22] Z. Guo, R. G. Melhem, R. W. hall, D. M. Chiarulli, and S. P. Levitan, "Pipelined communications in optically interconnected arrays," *Journal of Parallel and Distributed Computing*, vol. 12, pp. 269–282, 1991.

[23] Z. Guo, "Optically interconnected processor arrays with switching capability," *Journal of Parallel and Distributed Computing*, vol. 23, pp. 314–329, December 1994.

[24] G. Zicheng, "Optically interconnected processor arrays with switching capability," *Journal of Parallel & Distributed Computing*, vol. 23, pp. 314–29, Dec. 1994.

[25] S. Pavel and S. G. Akl, "On the power of arrays with optical pipelined buses," in *Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'96), Vol. III* (H. Arabnia, ed.), (Sunnyvale, California, USA), pp. 1443–1453, August 1996.

[26] M. Hamdi, C. Qiao, and Y. Pan, "On the computing power of arrays of processors with optical pipelined buses," *Parallel Processing Letters*, vol. 8, pp. 503–513, Dec. 1998.

[27] M. Middendoft and H. ElGindy, "Matrix multiplication on processor arrays with optical buses," *Informatica*, vol. 22, pp. 255–62, Oct. 1998.

[28] A. Bourgeois and J. Trahan, "Relating two-dimensional reconfigurable meshes with optically pipelined buses," in *Proceedings 14th International Parallel and Distributed Processing Symposium (IPDPS 2000)*, (Los Alamitos, CA, USA), pp. 747–52, May 2000.

[29] A. Bourgeois and J. Trahan, "Relating two-dimensional reconfigurable meshes with optically pipelined buses," *International Journal of Foundations of Computer Science*, vol. 11, pp. 553–71, Dec. 2000.

[30] Y. Li, J. Tao, and S. Zheng, "A symmetric processor array with synchronous optical buses and switches," *Parallel Processing Letters*, vol. 8, no. 3, pp. 283–295, 1998.

[31] Y. Pan, "Order statistics on a linear array with a reconfigurable bus," *Future Generation Computer Systems*, vol. 11, pp. 321–328, June 1995.

[32] Y. Pan, "Basic data movement operations on the LARPBS model," in *Parallel Computing Using Optical Interconnections* (K. Li, Y. Pan, and S.-Q. Zheng, eds.), pp. 227–247, Kluwer Academic Publishers, 1998.

[33] J. L. Trahan, A. G. Bourgeois, Y. Pan, and R. Vaidyanathan, "Optimally scaling permutation routing on reconfigurable linear arrays with optical buses," *Journal of Parallel and Distributed Computing*, vol. 60, pp. 1125–1136, September 2000.

[34] J. L. Trahan, A. G. Bourgeois, Y. Pan, and R. Vaidyanathan, "Optimally scaling permutation routing on reconfigurable linear arrays with optical buses," in *Second Merged Symposium IPPS/SPDP, 13th International Parallel Processing Symposium & 10th Symposium on Parallel and Distributed Processing*, (San Juan, Puerto Roco), April 1999.

[35] M. Hamdi, "Enhanced mesh-connected computers for image processing applications," in *Proceedings. CAMP '95 Computer Architectures for Machine Perception (Cat. No.95TB8093). IEEE Comput. Soc. Press. 1995* (V. Cantoni, L. Lombardi, M. Mosconi, M. Savini, and A. Setti, eds.), (Como, Italy), pp. 375–383, Sept 1995.

[36] C. Qiao, "On designing communication-intensive algorithms for a spanning optical bus based array," *Parallel Processing Letters*, vol. 5, no. 3, pp. 499–511, 1995.

[37] M. Hamdi, J. Tong, and C. Kin, "Fast sorting algorithms on reconfigurable array of processors with optical buses," in *Proceedings. 1996 International Conference on Parallel and Distributed Systems (Cat. No.96TB100045). IEEE Comput. Soc. Press. 1996*, (Tokyo, Japan), pp. 183–188, June 1996.

[38] M. Hamdi and Y. Pan, "Communication-efficient algorithms on reconfigurable array of processors with spanning optical buses," in *Proceedings. Second International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN '96) (Cat. No.96TB100044). IEEE Comput. Soc. Press. 1996* (G.-J. Li, D. Hsu, S. Horiguchi, and B. Maggs, eds.), (Beijing, China), pp. 440–446, June 1996.

[39] Y. Mei and Chunming-Qiao, "Embedding binary trees in arrays with optical busses," in *Proceedings of the Fourth International Conference on Massively Parallel Processing Using Optical Interconnections* (J. Goodman, S. Hinton, T. Pinkston, and E. Schenfeld, eds.), (Montreal, Canada), June 1997.

[40] C. Qiao, "A unique design of fiber-optic interconnection networks and algorithms," in *Parallel Computing Using Optical Interconnections* (K. Li, Y. Pan, and S.-Q. Zheng, eds.), pp. 163–184, Kluwer Academic Publishers, 1998.

[41] C. Qiao and Y. Mei, "An improved embedding of binary trees in a square reconfigurable array with spanning optical buses," *Parallel Processing Letters*, vol. 8, no. 3, pp. 321–336, 1998.

[42] M. Hamdi, C. Qiao, Y. Pan, and J. Tong, "Communication-efficient sorting algorithms on reconfigurable array of processors with slotted optical buses," *Journal of Parallel and Distributed Computing*, vol. 57, pp. 166–187, May 1999.

[43] Y. Mei and C. Qiao, "An efficient embedding of binary trees in reconfigurable arrays with spanning optical buses," *International Journal of Distributed Systems & Networks*, vol. 2, no. 1, pp. 40–48, 1999.

[44] S. Pavel and S. Akl, "Integer sorting and routing in arrays with reconfigurable optical buses," in *Proceedings of 25th International Conference on Parallel Processing* (A. Bojanczyk, ed.), (Ithaca, NY, USA), pp. 90–94, Aug 1996.

[45] S. Rajasekaran and S. Sahni, "Sorting and routing on the array with reconfigurable optical buses," in *Proc. 1996 IEEE 2ed International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'96)*, (Singapore), June 1996.

[46] S. Pavel and S. Akl, "Integer sorting and routing in arrays with reconfigurable optical buses," in *Proceedings of the International Conference on Parallel Processing*, (Bloomingdale, Illinois), pp. 90–94, August 1996. volume 2.

[47] S. Pavel and S. Akl, "Efficient algorithms for the hough transform on arrays with reconfigurable optical buses," in *Proceedings of the International Parallel Processing Symposium*, (Maui, Hawaii), pp. 697–701, April 1996.

[48] S. Pavel and S. Akl, "Matrix operations using arrays with reconfigurable optical buses," *Journal of Parallel Algorithms and Applications*, vol. 8, pp. 223–242, 1996.

[49] S. Rajasekaran and S. Sahni, "Deterministic routing on the array with reconfigurable optical buses," *Parallel Processing Letters,*, vol. 7, pp. 219–224, Sept 1997.

[50] S. Rajasekaran and S. Sahni, "Computing on the array with reconfigurable optical buses," in *World Multiconference on Systemics, Cybernetics, and Informatics*, (Caracas, Venezuela), pp. 459–466, 1997.

[51] S. Rajasekaran and S. Sahni, "Sorting, selection, and routing on the array with reconfigurable optical buses," *IEEE Transactions on Parallel & Distributed Systems*, vol. 8, pp. 1123–32, Nov. 1997.

[52] S. Pavel and S. Akl, "Integer sorting and routing in arrays with reconfigurable optical buses," *International Journal of Foundations of Computer Science, Special Issue on Interconnection Networks*, vol. 9, pp. 99–120, March 1998.

[53] S. D. Pavel and S. G. Akl, "Computing the hough transform on arrays with reconfigurable optical busses," in *Parallel Computing Using Optical Interconnections* (K. Li, Y. Pan, and S.-Q. Zheng, eds.), pp. 205–226, Kluwer Academic Publishers, 1998.

[54] S. Rajasekaran and S. Sahni, "Fundamental algorithms for the array with reconfigurable optical busses," in *Parallel Computing Using Optical Interconnections* (K. Li, Y. Pan, and S.-Q. Zheng, eds.), pp. 185–204, Kluwer Academic Publishers, 1998.

[55] C.-H. Wu, S.-J. Horng, and H.-R. Tsai., "Template matching on arrays with reconfigurable optical buses," in *Operations Research and its Applications. Third International Symposium (ISORA'98)* (D.-Z. Du, X.-S. Zhang, and K. Cheng, eds.), (Kunming, China), Aug. 1998.

[56] H. Elgindy and S. Rajesekaran, "Sorting and selection on a linear array with optical bus system," *Parallel Processing LetterS*, vol. 9, pp. 373–383, Sept 1999.

[57] C.-H. Wu, S.-J. Horng, and H.-R. Tsai, "Efficient parallel algorithms for hierarchical clustering on arrays with reconfigurable optical buses," *Journal of Parallel & Distributed Computing*, vol. 60, pp. 1137–53, Sept 2000.

[58] C.-H. Wu, S.-J. Horng, H.-R. Tsai, J.-F. Lin, and T.-L. Lin, "An optimal parallel algorithm for computing moments on arrays with reconfigurable optical buses," in *Proc. 14th International Parallel and Distributed Processing Symposium (IPDPS 2000)*, (Los Alamitos, CA, USA), pp. 741–6, May 2000.

[59] C.-H. Wu, S.-J. Horng, Y.-R. Wang, and L.-G. Jeng, "Parallel algorithms for vector median filtering," in *Proceedings of the 4th International Conference on Algorithms and Architectures for Parallel Processing*, pp. 240–51, 2000.

[60] C.-H. Wu and S.-J. Horng, "L/sub 2/ vector median filters on arrays with reconfigurable optical buses," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 12, pp. 1281–92, Dec. 2001.

[61] C.-H. Wu and S.-J. Horng, "Scalable and optimal speed-up parallel algorithms for template matching on arrays with reconfigurable optical buses," *International Journal of Foundations of Computer Science*, vol. 14, no. 1, pp. 79–98, Feb. 2003.

[62] C.-H. Wu and S.-J. Horng, "Fast and scalable selection algorithms with applications to median filtering," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, pp. 983–92, Oct. 2003.

[63] Y. Pan and K. Li, "Linear array with a reconfigurable pipelined bus system — concepts and applications," in *Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'96), Vol. III* (H. Arabnia, ed.), (Sunnyvale, California, USA), pp. 1431–1441, August 1996.

[64] R. Roldan and B. J. d'Auriol, "A preliminary feasibility study of the LARPBS optical bus parallel model," in *Proceedings of the 17th International Symposium on High Performance Computing Systems and Applications (HPCS 2003) and OSCAR Symposium* (D. Senechal, ed.), (Sherbrooke, Quebec, Canada), pp. 181–188, May 2003.

[65] H. Kimm, "Inversion number algorithm on a linear array with a reconfigurable pipelined bus system," in *Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'96), Vol. III* (H. Arabnia, ed.), (Sunnyvale, California, USA), pp. 1398–1407, August 1996.

[66] Y. Pan, K. Li, and S.-Q. Zheng, "Fast nearest neighbor algorithms on a linear array with a reconfigurable pipelined bus system," in *Proceedings. Third International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN '97) (Cat. No.97TB100209). IEEE Comput. Soc. 1997* (F. Lai, B. Maggs, and D. Hsu, eds.), (Taipei, Taiwan), pp. 444–450, Dec 1997.

[67] B. Cong, "On embeddings of neural networks into massively parallel computer systems," in *Proceedings of the IEEE 1997 National Aerospace and Electronics Conference. NAECON 1997.*, (Dayton, OH), pp. 231–238, July 1997.

[68] J. L. Trahan, Y. Pan, R. Vaidyanathan, and A. G. Bourgeois, "Scalable basic algorithms on a linear array with a reconfigurable pipelined bus system," in *10th International Conference on Parallel and Distributed Computing Systems*, (New Orleans, LA, USA), pp. 564–569, 1997.

[69] H. Kimm, "Parallel total weight crossing number algorithm for channel routing on a linear array with a reconfigurable pipelined bus system," in *Proceedings of the Twenty-Ninth Southeastern Symposium on System Theory (Cat. No.97TB100097). IEEE Comput. Soc. Press. 1997*, (Cookeville, TN, USA), pp. 183–187, March 1997.

[70] K. Li, "Boolean matrix multiplication on a linear array with a reconfigurable pipelined bus system," in *Proc. of the 11th Annual International Symposium on High Performance Computing Systems (HPCS'97)*, (Winnipeg, Manitoba, Canada), pp. 179–190, July 1997.

[71] K. Li, "Constant time boolean matrix multiplication on a linear array with a reconfigurable pipelined bus system," *Journal of Supercomputing*, vol. 11, no. 4, pp. 391–403, 1997.

[72] K. Li, Y. Pan, and S.-Q. Zheng, "Scalable parallel matrix multiplication using reconfigurable pipelined optical bus systems," in *Proceedings of the 10th IASTED International Conference on Parallel and Distributed Computing and Systems* (Y. Pan, S. G. Akl, and K. Li, eds.), (Las Vegas, Nevada, USA), pp. 238–243, Oct. 1998.

[73] K. Li, Y. Pan, and S.-Q. Zheng, "Fast and efficient parallel matrix computations on a linear array with a reconfigurable pipelined optical bus system," in *Proceedings of HPCS '98: 12th Annual International Symposium on High Performance Computing Systems* (J. Schaeffer, ed.), (Calgary, Alta., Canada), pp. 363–380, May 1998.

[74] Y. Pan, M. Hamdi, and K. Li, "Efficient and scalable quicksort on a linear array with a reconfigurable pipelined bus system," *Future Generations Computer Systems*, vol. 13, pp. 501–513, May 1998.

[75] Y. Pan and K. Li, "Linear array with a reconfigurable pipelined bus system. concepts and applications.," *Information Sciences*, vol. 106, pp. 237–258, May 1998.

[76] K. Li, "Fast matrix multiplication and related operations using reconfigurable optical busses," in *Parallel Computing Using Optical Interconnections* (K. Li, Y. Pan, and S.-Q. Zheng, eds.), pp. 248–273, Kluwer Academic Publishers, 1998.

[77] K. Li, Y. Pan, and S. Q. Zheng, "Fast and processor efficient parallel matrix multiplication algorithms on a linear array with a reconfigurable pipelined bus system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, pp. 705–720, August 1998.

[78] Y. Pan, K. Li, and S.-Q. Zheng, "Fast nearest neighbor algorithms on a linear array with a reconfigurable pipelined bus system," *Parallel Algorithms and Applications*, vol. 13, pp. 1–25, 1998.

[79] K. Li, Y. Pan, and S. Zheng, "Parallel matrix computations using a reconfigurable pipelined optical bus," *Journal of Parallel & Distributed Computing*, vol. 59, pp. 13–30, Oct 1999.

[80] Y. Han, Y. Pan, and H. Shen, "Fast parallel selection on the linear array with reconfigurable pipelined bus system," in *Proceedings. Frontiers '99. Seventh Symposium on the Frontiers of Massively Parallel Computation. IEEE Comput. Soc. 1999*, (Annapolis, MD, USA), pp. 286–293, Feb 1999.

[81] K. Li and V. Pan, "Parallel matrix multiplication on a linear array with a reconfigurable pipelined bus system," in *Proceedings 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing. IPPS/SPDP 1999*, (San Juan, Puerto Rico), pp. 31–35, April 1999.

[82] K. Li, Y. Pan, and M. Hamdi, "Solving graph theory problems using reconfigurable pipelined optical buses," in *Parallel and Distributed Processing. 11th IPPS/SPDP'99 Workshops held in conjunction with the 13th International Parallel Processing Symposium and the 10th Symposium on Parallel and Distributed Processing*, (San Juan, Puerto Rico), pp. 911–923, April 1999.

[83] K. Li, Y. Pan, and S. Zheng, "Efficient deterministic and probabilistic simulations of PRAMs on linear arrays with reconfigurable pipelined bus systems," *Journal of Supercomputing*, vol. 15, pp. 163–181, Feb 2000.

[84] B. J. d'Auriol, "Communication in the LARPBS (optical bus) model: A case study," in *Proc. of The Fourth International Conference on Algorithms And Architecture for Parallel Processing (ICA3PP2000)* (A. G. et al., ed.), (Hong Kong), pp. 581–590, December 2000.

[85] A. Datta and S. Soundaralakshmi, "Computing connected components on a linear array with a reconfigurable pipelined bus system," in *Proceedings of PART2000. Seventh Australasian Conference on Parallel and Real-Time Systems* (H. ElGindy and C. Fidge, eds.), (Sydney, NSW, Australia), pp. 181–191, Nov 2000.

[86] K. Li, Y. Pan, and M. Hamdi, "Solving graph theory problems using reconfigurable pipelined optical buses," *Parallel Computing*, vol. 26, pp. 723–735, May 2000.

[87] M. He and S. Q. Zheng, "An optimal sorting algorithm on a linear array with reconfigurable pipelined bus system," in *PDCS 2002: 15th International Conference on Parallel and Distributed Computing Systems*, pp. 386–91, 2002.

[88] A. Datta and Soundaralakshmi, "Fast and scalable algorithms for the euclidean distance transform on the larpbs," in *Proceedings of IPDPS Workshop on Advances in Parallel and Distributed Computational Models*, (San Francisco), April 2001.

[89] K. Li and V. Y. Pan, "Parallel matrix multiplication on a linear array with a reconfigurable pipelined bus system," *IEEE Transactions on Computers*, vol. 50, pp. 519–525, May 2001.

[90] A. Datta, "Efficient graph algorithms on a linear array with a reconfigurable pipelined bus system," in *Proceedings of IEEE International Symposium on Parallel and Distributed Processing*, (San Francisco, CA, USA), April 2001.

[91] K. Li, "Fast and scalable parallel algorithms for matrix chain product and matrix powers on reconfigurable pipelined optical buses," *Journal of Information Science and Engineering*, vol. 18, pp. 713–727, Sept 2002.

[92] Y. Pan, Y. Li, J. Li, K. Li, and S. Zheng, "Efficient parallel algorithms for distance maps of 2d binary images using an optical bus," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 32, pp. 228–236, March 2002.

[93] Y. Pan, Y. Li, J. Li, K. Li, and S. Zheng, "Efficient parallel algorithms for distance maps of 2d binary images using an optical bus," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 32, pp. 228–236, March 2002.

[94] A. Datta, "Efficient graph-theoretic algorithms on a linear array with a reconfigurable pipelined bus system," *Journal of Supercomputing*, vol. 23, pp. 193–211, Sept 2002.

[95] A. Datta, S. Soundaralakshmi, and R. Owens, "Fast sorting algorithms on a linear array with a reconfigurable pipelined bus system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, pp. 212–222, March 2002.

[96] Y. Han, Y. Pan, and H. Shen, "Sublogarithmic deterministic selection on arrays with a reconfigurable optical bus," *IEEE Transactions on Computing*, vol. 51, pp. 702–707, June 2002.

[97] J. Li, Y. Pan, and H. Shen, "More efficient topological sort using reconfigurable optical buses," *The Journal of Supercomputing*, vol. 24, pp. 251–258, March 2003.

[98] H. Kimm and D. Semé, "Longest repeated suffix problem on the arrays with pipelined optical bus systems," in *Proceedings of the 35th Southeastern Symposium on System Theory*, (Morgantown, WV, USA), pp. 69–73, March 2003.

[99] Y. Pan, "Computing on the restricted LARPBS model," in *Proc. of The 2003 International Symposium on Parallel and Distributed Processing and Applications*, (Aizu-Wakamatsu City, Japan), pp. 9–13, July 2003. Lecture Notes in Computer Science, vol. 2745.

[100] A. G. Bourgeois and J. L. Trahan, "Fault tolerant algorithms for a linear array with a reconfigurable pipelined bus system," *Parallel Algorithms and Applications*, vol. 18, pp. 139–153, September 2003.

[101] L. Chen, Y. Pan, and X. Xu, "Scalable and efficient parallel algorithms for euclidean distance transform on the LARPBS model," *IEEE Transactions on Parallel and Distributed Systems, to appear*, 2004.

[102] L. Chen, H. Chen, Y. Pan, and Y. Chen, "A fast efficient parallel hough transform algorithm on LARPBS," *The Journal of Supercomputing*, vol. 29, pp. 185–195, 2004.

[103] B. J. d'Auriol and R. Molakaseema, "A parameterized linear array with a reconfigurable pipelined bus system: LARPBS(p))," *The Computer Journal*, vol. 48, no. 1, pp. 115–125, 2005.

[104] A. Datta and S. Soundaralakshmi, "Fast and scalable algorithms for the euclidean distance transform on a linear array with a reconfigurable pipelined bus system," *Journal of Parallel and Distributed Computing*, vol. 64, no. 3, pp. 360–369, 2004.

[105] Y. Li, Y. Pan, and S. Zheng, "Pipelined time-division multiplexing optical bus with conditional delays," *Optical Engineering*, vol. 36, pp. 2417–2424, September 1997.

[106] J. Fernandez-Zepeda, R. Vaidyanathan, and J. Trahan, "Improved scaling simulation of the general reconfigurable mesh," in *Proceedings. Parallel and Distributed Processing, 11 IPPS/SPDP'99 Workshops*, (San Juan,Puerto Rico), pp. 615–624, April 1999.

[107] J. Fernandez-Zapeda, R. Vaidyanathan, and J. Trahan, "Using bus linearization to scale the reconfigurable mesh," *Journal of Parallel and Distributed Computing*, vol. 62, pp. 495–516, April 2002.

[108] B. J. d'Auriol and M. Beltran, "Optical bus communication modeling and simulation," in *Proc. of the International Symposium on High Performance Computing Systems and Applications (HPCS 2004)* (M. R. Eskicioglu, ed.), (Winnipeg, Manitoba, Canada), pp. 135–142, May 2004.

Edited by: Marcin Papzycki
Received: November 27, 2003
Accepted: May 28, 2004

AIMS AND SCOPE

The area of scalable computing has matured and reached a point where new issues and trends require a professional forum. SCPE will provide this avenue by publishing original refereed papers that address the present as well as the future of parallel and distributed computing. The journal will focus on algorithm development, implementation and execution on real-world parallel architectures, and application of parallel and distributed computing to the solution of real-life problems. Of particular interest are:

Expressiveness:
- high level languages,
- object oriented techniques,
- compiler technology for parallel computing,
- implementation techniques and their efficiency.

System engineering:
- programming environments,
- debugging tools,
- software libraries.

Performance:
- performance measurement: metrics, evaluation, visualization,
- performance improvement: resource allocation and scheduling, I/O, network throughput.

Applications:
- database,
- control systems,
- embedded systems,
- fault tolerance,
- industrial and business,
- real-time,
- scientific computing,
- visualization.

Future:
- limitations of current approaches,
- engineering trends and their consequences,
- novel parallel architectures.

Taking into account the extremely rapid pace of changes in the field SCPE is committed to fast turnaround of papers and a short publication time of accepted papers.

INSTRUCTIONS FOR CONTRIBUTORS

Proposals of Special Issues should be submitted to the editor-in-chief.

The language of the journal is English. SCPE publishes three categories of papers: overview papers, research papers and short communications. Electronic submissions are preferred. Overview papers and short communications should be submitted to the editor-in-chief. Research papers should be submitted to the editor whose research interests match the subject of the paper most closely. The list of editors' research interests can be found at the journal WWW site (`http://www.scpe.org`). Each paper appropriate to the journal will be refereed by a minimum of two referees.

There is no a priori limit on the length of overview papers. Research papers should be limited to approximately 20 pages, while short communications should not exceed 5 pages. A 50–100 word abstract should be included.

Upon acceptance the authors will be asked to transfer copyright of the article to the publisher. The authors will be required to prepare the text in $\LaTeX\,2_\varepsilon$ using the journal document class file (based on the SIAM's `siamltex.clo` document class, available at the journal WWW site). Figures must be prepared in encapsulated PostScript and appropriately incorporated into the text. The bibliography should be formatted using the SIAM convention. Detailed instructions for the Authors are available on the PDCP 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.