FMOODS/DAIS 2003 Student Workshop

Proceedings

Formal Methods for Open Object-based Distributed Systems
and Distributed Applications and Interoperable Systems

18th November 2003
Paris, France

http://fedconf.enst.fr/

Workshop Committee

Workshop Organisers:

Philip Greenwood
Computing Department, Lancaster University p.greenwood@lancaster.ac.uk

Kevin Lee
Computing Department, Lancaster University leek@comp.lancs.ac.uk

Matthias Zenger
Swiss Federal Institute of Technology
matthias.zenger@epfl.ch

Lynne Blair – FMOODS Liaison
Computing Department, Lancaster University lb@comp.lancs.ac.uk

Programme Committee:

FMOODS/DAIS 2003 Student Workshop

The combined "Formal Methods for Open Object-based Distributed Systems and Distributed Applications and Interoperable Systems" workshop is intended for PhD students working within the area of formal method support for Open Object-based distributed systems and technologies/platforms for reconfigurable, scalable, and adaptable distributed systems. The topics of the workshop include but are not restricted to:

- Formal models and techniques for specification, design or analysis
- Testing, validation and verification
- Formal support for software development
- Frameworks for modelling, specifying, monitoring and managing context-aware applications
- Support for reconfiguration, self-organisation and autonomic behaviour
- Foundations and applications of web services
- Formal models for coordination, components and component based software
- Semantics of object-oriented, component-oriented and aspect-oriented programming languages and systems

PhD Workshop Session 1 (Session Chair – Phil Greenwood)	9.00-12.45
• "Innovative Concept of Generic System Supervision" A. Sadovykh • "Operational ASM Semantics behind Graphical SEAM Notation" I. Rychkova, A. Wegmann, P Balabko • "Object-Oriented Graph Grammars" A. Ferreira • "Assembling Contracts for Components" F. Legond-Aubry, D. Enselme, G. Florin	
Coffee Break (10.45-11.05)	
PhD Workshop Session 2 (Session Chair – Kevin Lee)	
• "Enabling Re-Configurability on Component-based Programmable Nodes" J.Ueyama et al • "Why is Service-Orientation Necessary for Event Correlation?" A. Hanemann, D. Schmitz • "Aspect Testing Framework" D. Hughes, P. Greenwood, L. Blair • "SEDAM: Service Discovery in MANETs exploiting Asymmetric Mobility Patterns" G. Treu	

Innovative Concept of Generic System Supervision

Andrey SADOVYKH

Laboratoire d'Informatique, Université Paris 6 (LIP6)
8, rue du Capitaine Scott, 75015 Paris, France
Andrey.Sadovykh@lip6.fr

Abstract. Growing complexity of distributed engineering, simulation and learning systems requires capability to effectively monitor and supervise remote components. Author's PhD research aims at developing a generic concept of supervision to achieve greater portability, interoperability and to make building of comprehensive supervision system as simple as possible. According to this goal and considering the state of the art in middleware and supervision technologies, an open, modular, agent-based middleware concept is proposed. The supervision middleware is based on Web-Services framework and uses XML-based language for inter-agent communication. SOAP protocol, as a basis of a transport layer, allows straightforward development of highly portable supervision system. Modular architecture provides capability to integrate third-party agents and to benefit from supervision at different levels such as Application, Network, System.

1 Introduction

The PhD research is performed in the frame of GeneSyS project (IST-2001-34162) co-funded by the Commission of the European Community (5th Framework). EADS SPACE Transportation (France) is the project Coordinator, with University of Stuttgart (Germany), MTA SZTAKI (Hungary), NAVUS GmbH and D-3-Group GmbH (both of Germany) as participants. GeneSyS was started in March 2002 and is planned to be completed in September 2004 [1]. Although the final concept and implementation aspects as revealed in the article are the results of a tight collaboration between all the involved researchers, author contributed to the conception, implementation and results evaluation as a representative of EADS-ST. The contribution is emphasised in the concept and implementation sections.

1.1 Motivation

There are several supervision frameworks currently available such as SNMP [2], JMX [3] and Corba [4], which are used in various supervision systems like Tivoli (from IBM), OpenView (from HP) and NAGIOS. They are aimed at different aspects of monitoring starting from operating systems through network and up to applications.

However most of them have several constraints, some of them are mentioned in the list below:

- *Interoperability issues:* Components written on different languages using different toolkits, which are supposed to use the same architecture specification , may not be capable to co-operate full scale.
- *Components portability:* Often components are build to work only under their native operating systems like MS-Windows or Linux. They are very sensible to transport mechanism and more generally speaking to low level communication protocols.
- *Development/deployment complexity:* Many commercial applications have proprietary APIs that makes it difficult to create new agents and plug them to existing supervision system.
- *Non-flexible architecture:* When agent and visualisation tools are released in the same component, upgrades of console impact agent functionality and visa versa.
- *Dedication to a particular monitoring layer*, *lack of comprehensive solutions:* For instance, there exist various application layer tools to supervise Oracle database. It would be very useful to get simultaneously system information and network statistics to better control the system.

These constraints complicate integration between third- party monitoring tools to ensure system control on all the levels. In the meantime, proprietary solutions slow down pace of development of the whole domain.

The PhD research challenges to propose an open, generic modular and comprehensive supervision concept, which is due to solve or to alleviate the above mentioned problems, and to validate this concept on prototype implementation in various industrial contexts.

Following section intends to give an example of a real live industrial problem.

Preliminary Design Review.

The Preliminary Design Review (PDR) [12] is a major mile stone used in EADS-ST during the development of spacecrafts. The GeneSyS took the Automated Transfer Vehicle (ATV) PDR as a validation scenario (the real PDR has been performed traditional way). ATV is used to service and to reboost the ISS (International Space Station).

This Programme is led by EADS-ST as Prime contractor on behalf of ESA (European Space Agency). During the Program life-cycle several reviews are planed. The PDR is one of the first of them [12]. During a PDR, engineers from different countries collaborate on ATV design documentation, create Review Items Discrepancies (RID), meet on-line to discuss RIDs and possibly release Change Proposals. To support these activities a Distributed Engineering system, called PDR application, is applied.

This system consists of a document repository, called Engineering Database (EDB) server, and video conference server which routes H.323 and T.120 protocol streams and manages sessions (on-line meetings). The EDB server contains thousands of documents. Both servers support up to one hundred users simultaneously. Review meetings involves up to fifty engineers.

To efficiently manage the PDR process, a specific supervision solution is required. The supervision is needed for operating systems, network and application involved.

Although the servers are hosted on Linux, client applications, documentation front-end and video chat, are run under Windows. The network is heterogeneous, since reviewers are located in different countries and they use various server access means, including ISDN and Ethernet networks. Moreover, some client hosts are hidden by firewalls and network address translation. As for application layer, EDB is built using Java e-business solutions and conference server developed on C++ and uses PHP based application for session management. These constraints shape the supervision solution.

1.2 State of the Art

This section lists available technologies and points at some of the researches held in the domain of the distributed supervision.

Current supervision technologies:

SNMP: Simple Network Management Protocol [2] is the most widely used protocol for the management of IP based networks. Its concept also allows management of end systems and applications using specific Agents and Management Information Bases.

FIPA: The Foundation for Intelligent Physical Agent [5] is a non-profit international organisation that promotes the industry of intelligent agents by developing specifications supporting interoperability among agents and agent-based applications. These specifications include Abstract Architecture and FIPA Agent Communication Language.

JMX: Java Management Extension [3] is a SUN specification describing the design patterns of smart Java agents for application and network management. The JMX propose a three-layer architecture comprising Instrumental level, Agent level and Distributed Services level.

OMG: *Corba and Corba Component Model technologies* [4] have embedded distributed objects management service, which is highly used in the modern supervision frameworks like Tivoli (IBM).

W3C - Web Services: The Web Services is an E-business technology [8] for world-wide applications and based on SOAP, WSDL, UDDI [6], XML-based protocols for service interaction, description and lookup. Thus this can be used for supervision data exchange. For example, WSDM Technical Comity, tries to define web services management of distributed resources. The work is ongoing.

The following table outlines a comparison of the most important parameters to achieve the declared goals.

Table 1. Middleware technologies comparison

Criteria:	*FIPA*	*CORBA*	*JMX*	*Web Services*
Portability:				
Protocol	Binary: Proprietary,	Binary: GIOP	Binary: RMI or Servlets over	XML based :SOAP

	TCP/IP based		HTTP	
Languages	C/C++, Java	Many languages. Depending on realisation	Java	Many languages. Depending on realisation
Operating Systems	Windows, Linux, Unix	Windows, Linux, Unix	Java enabled platforms	Windows, Linux, Unix
Commodity :				
Services:	Embedded Directory and Naming services, Repository	More than 5, including Directory, Naming, Transactions etc. services	Directory and Naming services	Capability to plug Directory and Naming service.
Number of realisations	More than 5	More than 10	Sun	More than 20
Development complexity	Medium	High	Medium	Medium
Applicability to PDR scenario:				
Operating Systems	yes	yes	yes	yes
Network Protocols	yes	yes	yes	yes
Firewall Transparency	no	no	no	yes
Programming Languages	No support for PHP	No support for PHP	No support for PHP, C/C++	yes
Other points	C/C++, Java toolkits are not interoperable	Difficult to design and deploy	Great support for E-business development	E-business technology

Relevant Research Projects [7]:

Supervision related projects: ANDROID, MANTRIP, SHUFFLE projects are mostly aimed at network supervision. There are also projects related to system resource supervision like AgentScape and OPENDREAMS.

Agents management related projects: AgentLight and LEAP projects are dedicated to the mobile platforms based on J2ME [3] and FIPA. AgentCities project proposes FIPA world-wide network. SAFIRA project address real-time multi-agent middleware domain.

Intelligent agents related projects: Agent Academy and RACING projects are concentrated on a data-mining framework for intelligent agent. PISA project deals with development of security software agents for the Internet and E-commerce.

2 Concept

The state of the art shows that there is no off-the-shelf solution that fits all the requirements as expressed earlier in this document. However, existing paradigms can be applied to build a new generic supervision concept.

Analysing and synthesising the current supervision frameworks, the basic concept of the present research work has been determined. This concepts is also one of the main outcome and foundation stone of the GeneSyS project. The final, more detailed, concept has been developed by mapping this concept to possible technical solutions. Therefore, functional services and its physical implementation has been distinguished that emphasises the fact that the concept is not linked to a particular technology.

2.1 General Concept

The basic functionality that generic supervision solution should have are expressed here-by:

- The supervision system should be simple to connect and to extend it with advanced capabilities (3d party tools, summarisers, intelligent agents, consoles).
- Components should be portable and easy to develop, using well-known, standard technologies and patterns (development guidelines).
- High level of flexibility requires separation of monitoring tools from visualisation application and communication bus (3-layer architecture).
- The system should have relative transparency on the protocol and programming API levels (like CORBA GIOP or W3C SOAP).
- Monitoring tools or information should be described in standard and programming language independent manner (meta-language like CORBA IDL or W3C WSDL).
- The data source lookup, the data persistency and notification services should be provided (like CORBA services or UDDI).

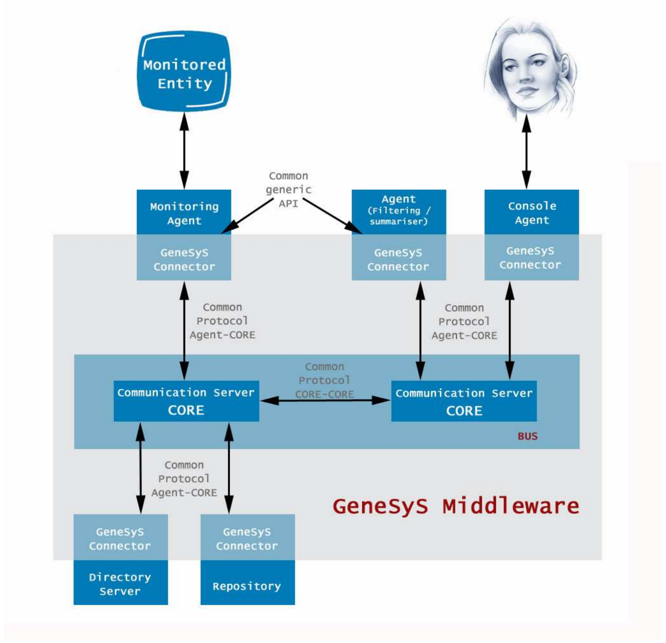

Fig. 1. High level overview of basic architecture

According to the previous statements, Figure 1 depicts general building blocks and components of the basic architecture as follows:

- *Monitoring Entity* provides monitoring information to the Agent. It can be SNMP MIB, Linux proc file system or even an API, for example Java Media Framework.
- *Monitoring Agent* implements an interface to the entity which is being monitored. Depending on its purpose, it stores monitoring data locally or forward it further to other agents via a Connector and Communication Bus (Central box in the Fig. 1).
- *Console Agent* is also plugged to Communication Bus via a Connector. It should process the data received from other agents and visualise it in a user-friendly form. One of the developers goals is to design a data massage format to allow building of an universal console that can represent all type of data, Generic Console.
- *Connector* plugs agents to the Communication Bus and provides for communication transparency. Connector can be a specific transport layer API, portable adapter or a protocol. Thus, it help in development of new agents simplifying communication with the Communication Bus.
- *Communication Bus (Communication Server, Core)* is a data transport facility comprising message formats, network protocols, databases and specialised servers. Communication Bus keeps agent locations and descriptions on a Directory Server. For data persistency, it can also store the messages in a Repository.
- *Directory Server* is to provide with publish/subscribe and look-up services.
- *Repository* is purposed to log collected data.

According the requirements, the Communication bus should have a platform independent implementation and manage data exchange in a standardised manner. Certain flexibility should be achieved by separating visualisation facilities from the instrumented applications connected to monitored entity.

The Agents have been classified as *Producers* (Monitoring agents) and *Consumers* (Console agents). The Complex Agents implement both Producer and Consumer paradigms. For example, providing summarisation, data analysis, proxy capability, the agent should be a consumer for some kind of information and should produce another. For higher flexibility and standardisation, the Directory Server and Repository implement the Agent paradigm too.

All middleware technologies mentioned earlier can be used to build the required system. The application management requires special portability of the components. Extending existing application with supervision can be quite difficult. From our point of view, Web Services have necessary and cost effective means and could be used in the transport layer of a new supervision architecture.

2.2 Web-Services Implementation

As a mapping of the requirements onto Web-Services concept [8], the following architecture is proposed.

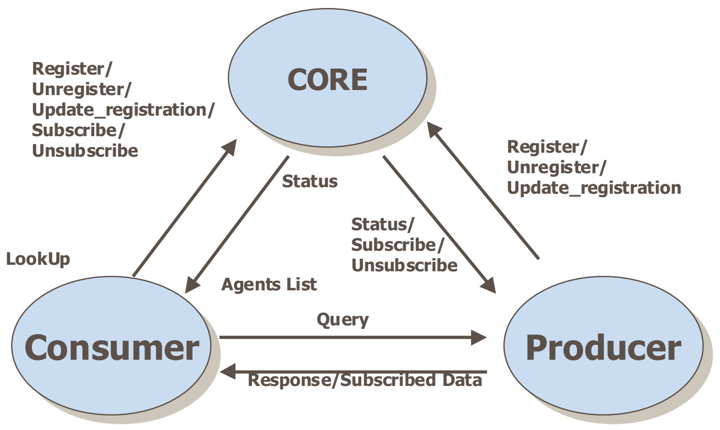

Fig. 2. Web-Services implementation of the generic supervision middleware.

In this picture, Consumer corresponds to Client of Web-Services concept and Consuming Agent of basic concept from previous section. Producer corresponds to Service and Producing Agent. The Core plays a role of directory and naming server. The Communication bus, in Web-Services implementation, is integration of the Core, Web-Services protocols and data message format. The Consumer and Producer register their location and abilities in the Core. The Consumer directly communicates with the Producer using Query/Response and Publish/Subscribe via SOAP protocol. Interfaces to the components are described in WSDL.

Our innovation is specification of the monitoring and service messages using XML based language specific to problematic and similar to FIPA ACL. In our specification, the message includes a header part containing addresses and timestamp and a body part containing a monitoring or service message. The message types are described in XSL (XML Schema Language), which allows comprehensive description of all details. This message classification permits to find an agent not only by agent name or location but also by monitoring capability it provides. SOAP, as a transport, provides high level of portability. SOAP was ported to HTTP, FTP, SMTP and have some binary realisations for better performance. Adapters for Java RMI and Corba GIOP are available and allow co-operation with existing supervision frameworks.

Due to the use of WSDL mentioned in state of the art and XSL, the core can be easily extended with a global UDDI registry [6].

High level of standardisation allows building generic visualisation tools, Generic Console concept mentioned earlier.

2.3 Results So Far

Recently, the validation phase of the first prototypes has been finished. Agents for monitoring operating system, network, databases, GroupWare applications were validated within the domain of distributed engineering in space industry. Moreover, agents were developed using Java AXIS, C++ gSOAP and .NET toolkits and NuSOAP for PHP, that proved a great portability of the chosen solution. Generic and application specific monitoring consoles have been used to perform world wide validation scenario involving international partners. Distributed PDR scenario has been played with GeneSyS partners spread in France, Germany, Hungary and Russia. Evaluation comments of real users have been gathered and exploited to allow improvement of the concept. A new version is planned for implementation.

The validation process proved that, although visualisation consoles still should be improved, the developed components provide valuable monitoring information.

2.4 Future Plans

New versions of prototypes will address control and security issues. The development will focus on visualisation tools, intelligent agents and co-operation with other supervision solutions. Astronauts training scenario will require design of new agents for real-time simulators monitoring, which will extend application monitoring subject.

3 Conclusions

The concept is innovative application of web-services technology. Proposed supervision framework can be useful in different kinds of distributed systems and scenarios such as distributed simulation [9], distributed system engineering [10] and distributed learning [11]. The article addresses various industrial contexts, modern technologies and research domains.

References

1. GeneSyS project official web-site: http://genesys.sztaki.hu
2. Simple Network Management Protocol http://www.snmp.org
3. Java Management Extension http://java.sun.com
4. Object Management Group http://www.omg.org
5. Foundation for Intelligent Physical Agent http://www.fipa.org
6. World Wide Web Consortium specifications http://www.w3c.org and UDDI organisation http://www.uddi.org
7. Agentlink project http://www.agentlink.org
8. Web Services organisation http://www.webservices.org
9. J-E Bohdanowicz, A. Laydier, P. Chliaev, A. Vankov, V. Voloshinov, A. Sadovykh: GeneSyS Project: Supervision of Distributed Systems (03E-SIW-043). EuroSIW 2003 Conference Proceedings, Stockholm, Sweden (2003).
10. A. Sadovykh, S. Wesner, J-E Bohdanowicz: "GeneSyS: A Generic Architecture for Supervision of Distributed Applications". EuroWeb 2002 Conference Proceedings, Oxford.
11. L. Arguello, A. Vankov, P. Chliaev, V. Voloshinov, V. Krivtsov, O. Estehin, A. Alyoshin, A. Vislotsky, A. Sadovykh: Distributed Learning with Online Simulations for ISS Payload Training. ESA/ESTEC Conference Proceedings, Noordwijk, The Netherlands (2002).
12. ECSS Space Engineering, System Engineering(ECSS-E-10A). ESA-ESTEC Requirements & Standard Division, Noordwijk, The Netherlands.

Operational ASM Semantics behind Graphical SEAM Notation

Irina Rychkova, Alain Wegmann[1], and Pavel Balabko[1]

School of Computer and Communication Sciences (I&C), École Polytechnique Fédérale de Lausanne (EPFL) CH-1015 Lausanne, Switzerland
{Irina.Rychkova, Alain.Wegmann, Pavel.Balabko}@epfl.ch
http://lamswww.epfl.ch

Abstract. The context of this paper is Enterprise Architecture (EA). EA is a multi-disciplinary approach that allows different specialists to design new business and IT systems and focuses on the integration of these systems. Our group develops a specific EA method that is called SEAM. The current version of SEAM has a formal denotational semantics for its modeling language. In order to provide model simulation and checking at each level of abstraction, SEAM needs an operational semantics. Currently this work is at the stage of problem setting. In this paper[1]we present SEAM and describe the main research problem. We propose to use ASM as operational semantics for SEAM to verify that models, produced by different specialists are consistent. We illustrate our approach by giving an example of SEAM notation that has already been mapped to ASM.

1 Introduction

Enterprise architecture (EA) is a multi-disciplinary approach that enables enterprises to anticipate or react to necessary business or technical changes. The EA team designs and deploys new organizations and IT systems in the light of necessary changes. In an EA project, the EA team develops a model that represents the enterprise: the enterprise model. The enterprise models are usually structured in hierarchical levels. The highest level describes the marketing aspects, the middle level describes the business processes, and the lower level describes the IT systems.[1]

Our group develops a theory for enterprise architecture (EA) and then applies this theory to the development of a specific EA method called SEAM [1]. SEAM stands for the "Systemic Enterprise Architecture Methodology" or for seamless integration between business and IT.

The important parts of SEAM are *the method* and *the notation.* The SEAM method explains how to proceed in the analysis and design of the enterprise[1].

[1] This paper is written by Irina Rychkova. Alain Wegmann, the scientific advisor of the project, proposed to study the EA context. This work is a continuation of the work done by Pavel Balabko, who proposed to consider ASM in context of our research problem.

The SEAM notation defines a graphical modeling language. This work mainly deals with the SEAM notation. In context of EA, graphical notation is essential. Graphical model representations can be less ambiguous and much more efficient for communication than plain text. As EA projects involve specialists from different disciplines, it is important to provide means to simulate the model (to ease communication between the specialists) and means to check the model (to verify that the different levels developed by the different specialists are compatible). For this purpose, in addition to graphical notation which is defined by a denotational semantics, we need to provide an operational semantics for our modeling language.

In this paper we present our work that is currently at the stage of problem setting. Our project has 3 main goals: to define more precisely the SEAM modeling language, to provide an operational semantics for this language and to validate the impact of having an operational semantics in context of EA. For this paper we focused mostly on operational semantics for SEAM in order to provide model simulation and checking.

In section 2 of this paper we consider the SEAM method and its main aspects in the context of EA. In section 3 we formulate our research problem. In the first part of section 4 we justify the choice of ASM as a solution of our research problem and observe the advantages of this semantics. Then, in the second part of section 4, we consider how to define the SEAM notation in ASM: we discuss a tool we plan to develop and illustrate the modeling process on the example. Section 5 is the conclusion.

2 SEAM Method in Context of Enterprise Architecture

The goal of SEAM as a modeling language is to serve as a uniform notation for all enterprise stakeholders that participate in the modeling process. While developing the SEAM notation we are trying to be as close as possible to UML. At the same time, SEAM has several characteristics which, we believe, make it more appropriate than UML for EA modeling.

SEAM: A Method for Stepwise Design Using Hierarchical Models

SEAM enterprise model describes a hierarchy of systems, which includes the business and IT resources together with the processes in which they participate.
Hierarchical Model: The enterprise model is typically structured in levels (business, operational, IT)[2]. In SEAM, at each level, systems of interest can be represented with a computational viewpoint (CV), as a collaboration of subsystems. At the same time, each subsystem can be described with an information viewpoint (IV) (Fig. 1). This is inspired by RM-ODP.[7]

The computational viewpoint (CV): a viewpoint on the system of interest that enables distribution through the functional decomposition of the system into subsystems that interact at interfaces. CV is concerned with the description of the system as a set of physical objects.

The information viewpoint (IV): a viewpoint on the system of interest that focuses on the semantics of the information and information processing performed. IV is concerned with the information that needs to be stored and processed in the system and describes the behavioral aspects of the system.

As a result of the modeling process, we can build the hierarchy of CV specifications from whole to composite (collaboration of subsystems), (Fig. 1) and hierarchy of IV specifications from general to detailed (Fig. 2).
Stepwise design: SEAM method realizes recursive modeling of a hierarchical system. There is a universal modeling template that could be described as:

1. Take a CV of the system at the highest level (Fig. 1-(a) and Fig. 2-(a) that represent the same system "System1").
2. Make an IV specification for the sub-system(s) of interest. The IV specification(s) includes system policies and non-functional requirements documented as assumptions (Fig. 2-(b) that represents "subS1" sub-system of "System1" together with a set of assumptions).
3. Make a detailed IV specification for the sub-system(s) of interest. The detailed IV specifications transform the assumptions into behavior (Fig. 2-(c)that represents "subS1" information viewpoint with all necessary details for its implementation).
4. Define the CV at the next level by making a CV refinement of the system of interest (Fig. 1-(b) that represents "subS1" computational viewpoint corresponding to the information viewpoint of "subS1" in Fig. 1-(a)).
5. Make an IV specification for the subsystem(s) of interest (step 2).
6. Make a detailed IV specification for the sub-system(s) of interest (step 3).
7. Iterate steps 4 to 6 for all CV levels.
8. Verify that the model is complete and coherent.

It is interesting to highlight that SEAM is an evolution of Catalysis [11]. Some of the Catalysis originalities (that are kept in SEAM) are: the hierarchical system design (across 3 levels) and the concepts of joint actions to specify the system goals. SEAM keeps these features and adds a more precise ontology based on RM-ODP, the possibility to design more than 3 levels, and a notation better suited for system representation.

SEAM: A Notation for System Modeling

Modeling an enterprise across its levels is difficult. To make it practical, it is important to have a modeling notation well suited for system modeling. To illustrate this, we give one example which is the integration of the system's behavior and the system's information representation. Generally speaking, the UML diagrams can be categorized either as structural or as behavioral diagrams. However, system theory has shown that the separation behavior / structure is artificial [6] and actually prevents the development of models that truly represent the changes happening in the modeled system. For example, in UML, it is not possible to show in one diagram that an action changes the value of an attribute. That can make the diagram difficult to understand. This point has been

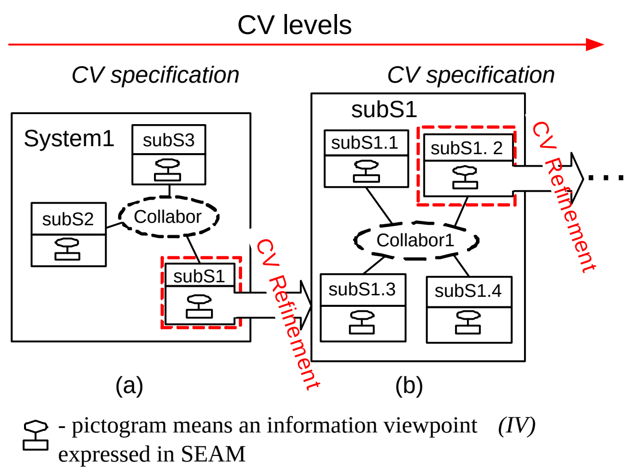

Fig. 1. SEAM hierarchy of subsystems. a) CV specifications of a System1 modeled as a collaboration of subsystems (composite view). Subsystem 1 (subS1) modeled as a whole. b)Subsystem 1 (subS1) is modeled as a composite (collaboration of subS1.1, subS1.2, e.t.c.).

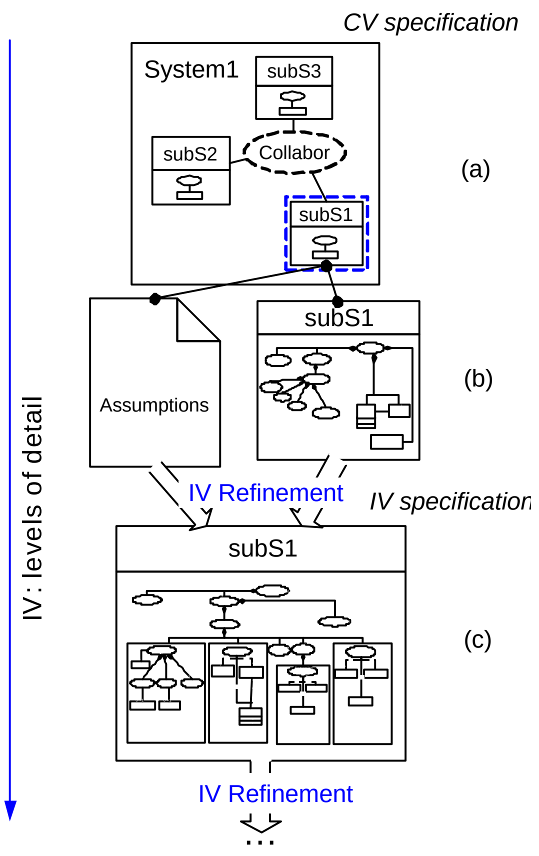

Fig. 2. SEAM levels of details. Top part of the figure shows the CV specification of the System1 that contains the very general IV specification of subsystem 1 (subS1). Middle and bottom parts of a figure show the result of subS1 model refinement. This set of IV specifications from general to detailed makes up a hierarchical model of system behavior.

identified by OPM language developers [4]. SEAM also proposes the solution for this problem.

3 Main Research Problem: Operational Semantics for SEAM Modeling Language

Denotational semantics and operational semantics, being transferred from the context of programming languages, play important roles in the definition of the modeling languages. *Denotational semantics* provide mathematical models and define relations between the terms of a modeling language. *Operational semantics* is essential for modeling languages when their applications are supposed to be simulated on a machine.

The current version of denotational semantics for the SEAM modeling language is based on [3]. In order to provide model simulation and checking, SEAM needs to have an operational semantics.

The possibility to *simulate* the model is the best way for people to figure out what is actually represented in the model. The possibility to *check* models is important for the comparison of models.

In our work we propose to map the SEAM notation into defined formal notation for which simulation and model checking tools have been developed. A good choice for these purposes is an Abstract State Machine (ASM) notation [5]. Note that the project will also have to define more precisely the SEAM modeling language and will evaluate the effect of the proposed approach by applying it to models existing in EA.

4 Abstract State Machines and SEAM Method

ASM is a method of stepwise refinable abstract operational modeling [5]. An ASM model can be used to capture the abstract structure and behavior of a discrete system.

4.1 Abstract State Machines as Operational Semantics for SEAM

In this work we propose to use ASM as an operational semantics for the SEAM modeling language. It is possible to talk about SEAM-ASM conformity for several reasons:

– In the context of requirements engineering it is important to have an abstract specification of a system without mention of its implementation. For users, in SEAM methodology a system can be represented with an information viewpoint (IV). At any level of details a SEAM IV specification can be described by an ASM specification.
– Both SEAM and ASM support principles of hierarchical system design. In ASM the hierarchy of intermediate models can be constructed by stepwise refinement (or adding more details) to the model. An ASM program can be

executed at any level of details. It corresponds to the SEAM hierarchy of IV specifications across level of details;

- In SEAM models we describe a system at any time as a pair (state, behavior), as well as in ASM. State is defined by a number of attributes and their current values. Behavior is defined as a set of actions that change a system state.

Using ASM as an operational semantics for SEAM we can obtain the following advantages for modeling:

Model Simulation. SEAM can represent models hierarchically, with several levels of detail (one of its benefits). Each IV specification in SEAM can be represented and simulated by ASM. (Fig. 3-a) This allows us to use ASM models as test models (to be matched by all stakeholders).

Refinement Checking. SEAM-ASM integration helps to make smooth and correct refinements. On each level we extend the system functionality and can also change its structure (redefine previous set of states). But each next level should correctly simulate the higher level model.(Fig. 3-b)

Model Validation and Version Comparison. ASM + verification tool = model validation and the comparison of alternative models, testing deadlocks and forbidden parameter combinations, generating of test sequences and possible sequences of states (functionality checking).(Fig. 3-c)

4.2 SEAM notation at ASM

We intend to develop an environment for our modeling techniques in order to gain practical benefits from its application. Environment can be divided into graphical, simulation, and verification tools.

Graphical tool. Graphical specifications are basic elements of our method. The Graphical tool is supposed to provide drawing and storing of SEAM models.

Simulation and verification tool. The ASM method has a tool support for simulation and verification of its models. Specifically, in [9] AsmL is presented. Another ASM tool environment for model simulation and verification is ASM Workbench [8]. In our tool we intend to use one of these tools by creating an interface or translator from SEAM to ASM notation.

Illustration of a Modeling Process (Vending Machine Example). To illustrate how our method works, we propose an example based on the Vending Machine (VM) case study [10] and its ASM specification, written in AsmL.[2]

[2] In this work we use an ASM-based specification language tool called Abstract State Machine Language or AsmL.[9] It is a language for modeling the structure and behavior of discrete dynamic systems based on the ASM method. AsmL specifications may be executed as programs. (tool support: AsmL tool version 2.0, developed by the Microsoft research group.)

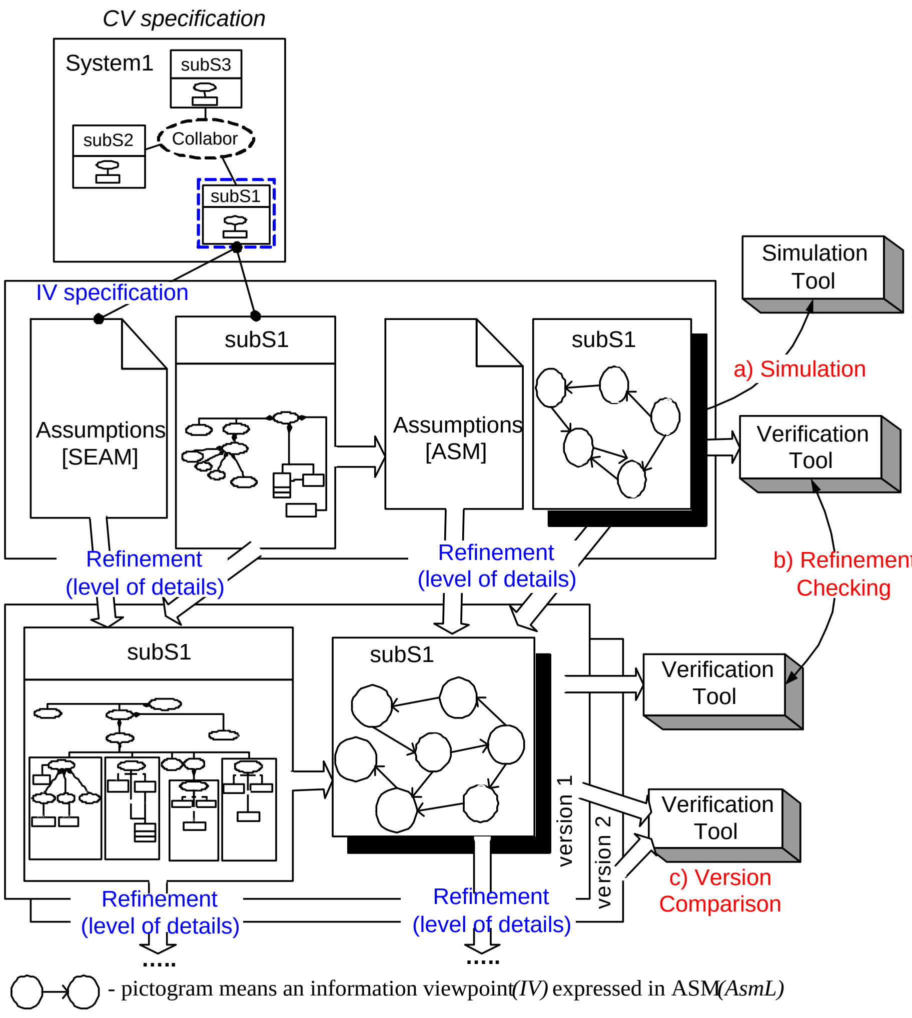

Fig. 3. ASM as an operational semantics for SEAM: a) The ASM method provides the simulation of SEAM models (IV specifications) on every level of details; b)using a verification tool for ASM models at different levels of details helps to make a correct model refinement; c) using a verification tool for different version of SEAM-ASM models allows us to make a model validation and version comparison.

VM Description. *A Vending Machine*(VM)[3] accepts money and, if there is sufficient credit, dispenses items selected by a customer. Items are identified by an alphanumeric code, called the selection number. Customers select items by entering the selection number via a keypad.

Studying the VM example we focused on the mapping between SEAM and ASM concepts. We started with an initial IV graphical specification for a model (Fig. 4), based on the VM description. Policies for VM operations were documented as assumptions. For example:

1. A product can be dispensed if a credit is sufficient.
2. A product can be dispensed if the machine has sufficient stock.
3. A change can be produced if there is enough money in the machine's reserve.
4. Invariant: sum of products (in cash equivalent) and the reserve should be constant for a particular machine during its service time.

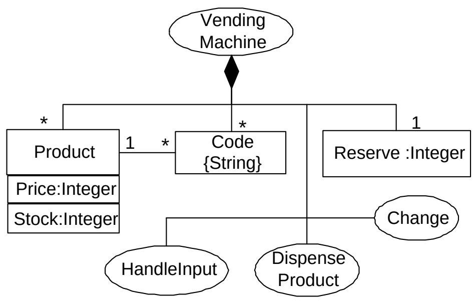

Fig. 4. IV Specification of VM. VM includes main concepts: Product, Code, and Reserve. Main actions (shown as ovals): HandleInput, DispenseProduct and Change.

In Fig. 5 we give the refined specification of the VM. New actions were added to explain the vending machine functionality. All the assumptions, made on the previous level, were transformed into actions with pre- and post- conditions. This specification can be easily mapped into the ASM notation (AsmL code in our case). Finally, an AsmL code was obtained[4] from the specification in Fig. 5. In this work the AsmL code was created by hand together with a rules for translation. Automatic AsmL code generation is a part of the future work.

For the Vending Machine we obtained the identical AsmL specification with the existed specification from the VM case study. As an illustration, below we propose an example of AsmL code for `SaleReady` operation (Fig. 5)

[3] In this work we simplified an original specification of VM [10]and reduced some requirements, such as restriction on the coin and bill denominations that the machine can accept and the ability to recognize coins and bills. Also in this example we assume that the customer always types the valid code of a product.

[4] AsmL Code for Vending Machine example is available at `http://lamspeople.epfl.ch/rychkova/Report%2008.07.2003/AsmLCode.pdf`

```
SaleReady() as Boolean
 return (Selection <> ['0','0']
         and not EmptyStock(ProductName())
         and Credit >= Price(ProductName())
         and not NoChange(Credit, Price(ProductName())))
```

Method `SaleReady` reflects the VM policies and returns "True" if no Null product is selected, selected product is available, credit is sufficient, and the machine has enough money to make change if necessary.

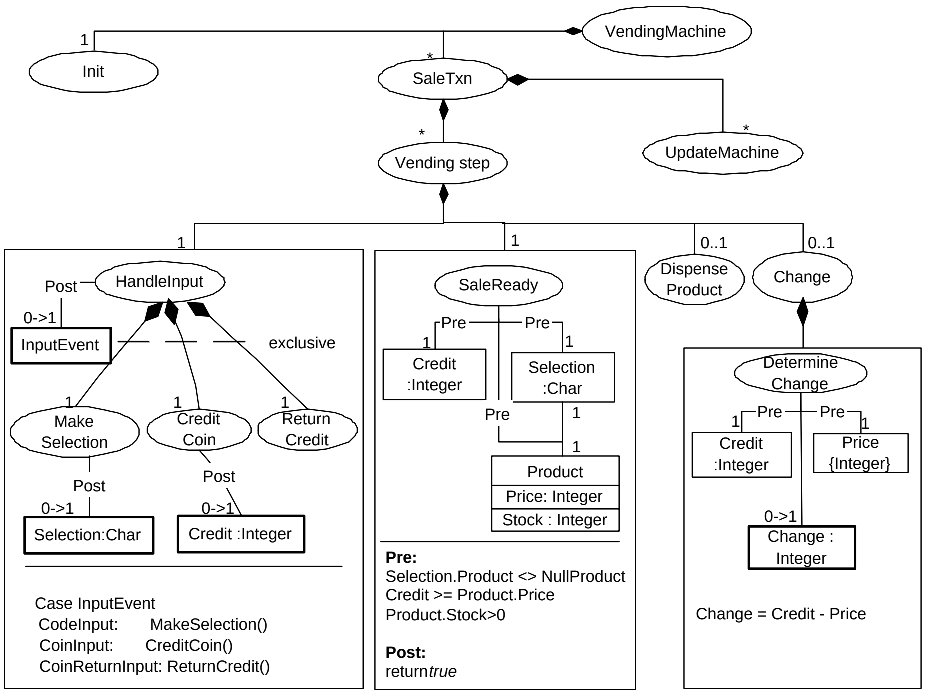

Fig. 5. Refined SEAM specification for VM. In this specification all necessary concepts, operations, and order of operations are shown. Necessary system policies formulated as a pre- and postconditions. AsmL code can be generated for each VM operation. Model can be read as following: Vending machine performance includes one `Init` operation and then set of `SaleTxn` (sale transaction) operations, where sale transaction can be considered as a dialog with one customer. Each sale transaction consists of a set of Vending steps (machine reaction on one input event) and may be finished by `UpdateMachine` operation. Each vending step can be described as a sequence of `HendleInput`, `SaleReady`, `DispenseProduct`, and `Change` operations.

As a result of this work, several basic rules for a mapping between the SEAM and ASM notations (namely, SEAM to AsmL translation) were generated.

5 Conclusion

We have presented an enterprise architecture (EA), an approach that allows a multi-disciplinary team to design enterprises (i.e. organizations and IT systems). To model complex systems with an EA team, it is important to have a unified modeling notation. For this purpose we propose the SEAM. However, to allow modelers to work more effectively, it is also important that models can be simulated and checked. This is a reason why we work on a mapping between our SEAM modeling language and ASM (for which simulation and model checking does exist).

We can highlight, that Egon Börger and Robert Stärk mention in [5] that ASM needs a graphical language that provides a "data model together with a functional model" to be usable with actual customers. With SEAM we provide such a notation.

References

1. Wegmann, A.: On the systemic enterprise architecture methodology (SEAM). Published at the International Conference on Enterprise Information Systems 2003 (ICEIS 2003), Angers, France.
2. Wegmann, A., Preiss, O.: MDA in Enterprise Architecture? The Living System Theory on the Rescue... Published at the 7th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2003) September 16-19 2003, Brisbane, Australia.
3. Naumenko, A., Wegmann, A., Atkinson, C.: The Role of Tarski's Declarative Semantics in the Design of Modeling Languages. Technical report.
4. Dori, D.: Object-Process Methodology - A Holistic Systems Paradigm. Springer Verlag, Berlin Heidelberg New York (2000)
5. Börger, E., Stärk, R.: Abstract State Machines. A Method for High-Level System Design and Analysis. Springer-Verlag, Berlin Heidelberg New York (2003)
6. Weinberg, M. W.: An Introduction to General System Thinking. Dorset House (2001).
7. Reference model of open distributed processing part 1. Draft International Standard (DIS), Helsinki, Finland, (15-18 May 1995)
8. Del Castillo, G.: The ASM Workbench. A tool environment for computer-aided analisis and validation of Abstract State Machine models. Dissertation. Fachbereich Matematik / Informatik und Heinz Nixdorf Institut Universitt Paderborn. Paderborn, 2000.
9. AsmL: The Abstract State Machine Language. Documentation prepared for Microsoft Research by Modeled Computation LLC, (2002) `http://www.modeled-computation.com`
10. Introducing AsmL: A tutorial for the Abstract State Machine Language. Vending Machine Case study. 2001, 2002 Microsoft Corporation. December 2001. `http://research.micosoft.com/foundations/AsmL`
11. D'Souza, D., Wills, A.C.: Objects, Components, and Frameworks with UML. The Catalysis Approach. Addison Wesley Longman, Inc. (1999)

Object-Oriented Graph Grammars

Ana Paula Lüdtke Ferreira

Centro de Ciências Exatas e Tecnológicas
Universidade do Vale do Rio dos Sinos – UNISINOS
anapaula@exatas.unisinos.br

Abstract This work aims to extend the algebraical approach to graph transformation to model object-oriented systems structures and computations. A graph grammar based formal framework for object-oriented system modeling is developed, incorporating both the static aspects of system modeling and the dynamic aspects of computation of object-oriented programs.

1 Introduction

The massive decrease on hardware costs disseminate the use of computers and computational devices within society. Consequently, different domains of application arise everywhere, and the more they are part of our daily lives, the more we depend on them to be correct regarding their intended behaviour. New software development techniques have emerged over the last years to deal with current needs, but the paradigms on which those techniques are based (especially objects, events, and concurrency) make testing and validation of systems more complex (and consequently, more error prone). This scenario requires specification techniques which can cope with the needs of modern software development. Such techniques must assure that the final product is consistent and complete regarding its specification, be formal, incremental and executable. Also, they must be simple enough to be used by non experts in formal methods.

Object-oriented development is perhaps the most popular paradigm of system development in use nowadays. Helped by the growing popularity of Java as a language to support WWW applications, object-oriented programs have been taken to a great spot of attention. Consequently, there has been an exponential increase of interest about models, semantics and verification of object-based systems.

There is a plethora of formalisms proposed in the literature to the specification of such systems. However, the object-orientation paradigm presents a number of idiosyncrasies, such as inheritance and dynamic binding, making those systems deviate considerably from others in both their architecture and execution. It should be expected that formalisms to specify them would reflect these idiosyncrasies, otherwise any attempt to use such formalisms for the specification of object-oriented architectures or programs is doomed to neglect key regularities in their organization. Only naturally, coherent specifications warrant the recognition of the underlying abstractions within the paradigm. This is equally true for object-oriented programs.

Object-oriented system modelling and programming approaches should present a number of desired properties, amongst which we cite the following: (i) the existence of a formal specification language which can be easily understood by both software developers and final users; (ii) the possibility of systems' static and dynamic aspects be specified in an integrated way; (iii) the existence of a semantical basis, allowing the composition of modular specifications in a consistent and significant manner; (iv) the possibility of high level specifications be refined into lower ones, or even into actual programs.

This paper is organized as follows: Section 2 presents the fundamental notions of graph grammars, and how they provide an adequate model of computation for distributed systems. Section 3 presents how the single pushout approach to graph grammars can be enhanced to describe object-oriented systems and programs. Finally, Section 4 presents some conclusions and future developments from the work presented here.

2 Graph Grammars

Graphs are a very natural way of describing complex situations on an intuitive level. Graph-based formal description techniques are, for that reason, easily used by non-specialists on formal methods. Graph transformation rules can be brought into those descriptions in order to enrich them with a model of computation, which can describe the evolution of a system represented as a graph.

The algebraic approach to graph grammars has been presented for the first time in [6] in order to generalize Chomsky grammars from strings to graphs. That approach is currently known as *double-pushout* approach, because derivations are based on two pushout constructions in the category [16][13] of graphs and total graph morphisms. The *single-pushout* approach [12], on the other hand, has derivations characterized as a pushout construction in the category of graphs and partial graph morphisms.

Generally, a graph grammar consists on an *initial graph* together with a finite set of *graph productions*. A graph production, or simply a *rule*, specifies how a system configuration may change. A rule has a *left-hand side* and a *right-hand side*, which are both graphs, and a partial graph morphism to determine what should be altered. Intuitively, a system configuration change occurs in the following way: all items belonging to the left-hand side must be present at the current state to allow the rule to be applied; all items mapped from the left to the right-hand side (via the graph morphism) will be preserved; all items not mapped will be deleted from the current state; and all items present in the right-hand side but not in the left-hand side will be added to the current state to obtain the next one.

Since rule application is non-deterministic, multiple rules can be applied at the same time, as long as there is no conflict [20] between them. This provides a true concurrency semantics for computations within that framework, which is especially adequate to model distributed systems.

Graph grammars are appealing as a specification formalism because they are formal, they are based on simple yet powerful concepts to describe behaviour, they have a nice graphical layout which helps the understanding (even by non-specialists in formal methods) of a specification. Since graph grammars also provide a model of computation, they can serve as the basis for specifications which can be executed on a computer.

3 Object-oriented graphs and grammars

The use of the object-oriented paradigm has increased over the past years, becoming perhaps the most popular paradigm of system development in use nowadays. The growing use of Java as a language to support Internet applications has also contributed to this popularity. Object-based systems have a number of advantages over traditional ones, such as ease of specification, code reuse, modular development and implementation independence. However, they also present difficulties, derived from the very same features that allow the mentioned advantages.

The most distinguished features of object-oriented systems are *inheritance* and *polymorphism*, which make them considerably different from other systems in both their architecture and model of execution. It should be expected that formalisms for the specification of object-oriented architectures or programs reflect these particularities, otherwise the use of such formalisms will neglect key concepts that have a major influence in their organization. According to [9], a specification language for object-oriented conceptual modeling must at least include constructs for specifying primitive objects, particularizations of predefined objects, inheritance relationships between objects and aggregation of objects in order to define more complex objects. We also believe that the concepts of polymorphism and dynamic binding are essential if we intend to model static *and* dynamic aspects of object-oriented systems. So, in order to correctly model object-oriented systems, the key concepts related to it must be present within the formalism used.

Graph grammars have been used in a variety of applications, and can provide suggestive and technically adequate models of computation, semantic foundations, and verification techniques. Different kinds of graph grammars have been proposed in the literature [12] [10] [5] [2] [17] [1] [19] [18] [3] [21] [23] [14] [22] [7], aiming the solution of different problems. However, the few focusing on object-oriented systems specification [15] [4] [11] do not present a treatment on inheritance and polymorphism, which make object-oriented systems analysis and testing so difficult.

This work aims to extend the algebraical approach to graph transformations to model object-oriented systems structures and computations. More accurately, the single pushout approach in the category of typed hypergraphs and partial typed hypergraph morphisms will be adapted to fit more adequately the object-oriented approach to software development. We will also show how the structures developed along the text are compatible with the notion of specification and computation within the object-oriented paradigm [8].

This extension is accomplished by realizing that inheritance is the construction which permits an object to be specialized from a pre-existing one, where the newly created object carries all the functionality from its primitive object. This relation induces a hierarchical relationship among the objects from a system, which can be viewed as a set of trees (single inheritance) or as an acyclic graph (multiple inheritance). Both structures can be formally characterized as a binary acyclic, functional relation with no reflexive pairs. The reflexive and transitive closures of it is a partial order relation, which can equip the set of nodes (objects) and edges (attributes and messages) of a (hyper)graph (a system, program, or part of it).

The core structure is called a *type hierarchy*, which consists of an (almost) ordinary hypergraph (called *object-model graph*), together with two binary acyclic, functional relation with no reflexive pairs, one relating vertices (the inheritance *isa* relation) and the other relating edges (the method redefinition relation) [8]. Since type hierarchies are algebraic structures, operations over them can be defined. *Composition* (done with or without identification of elements on the structures being composed) plays an important role, since it corresponds to system composition.

The formalism is based on hypergraphs typed over type hierarchies. Typing is achieved by means of typing morphisms (which are structure preserving mappings). These typing morphisms, however, take into consideration the *isa* relationship and also the overriding of methods, according to the object-oriented semantics: an arc can be incident to any node as long as it, and its typing edge are connected within the partial order generated by the *isa* relation. This definition reflects the idea that an object can use any attribute one of its primitive classes have, since it was inherited when the class was specialized.

Rules are restricted in order to capture some characteristics of the object-oriented paradigm: the left-hand side of a rule is required to contain exactly one element of type message, and this particular message must be deleted by the rule application, i.e., each rule represents an object reaction to a message which is consumed in the process. This demand poses no restriction, since systems may have many rules specifying reactions to the same type of message (non-determinism) and many rules can be applied in parallel if their triggers are present at an actual state and the referred rules are not in conflict. Systems' concurrent capabilities are so expressed by the grammar rules, which can be applied concurrently (accordingly to the graph grammar semantics), so one object can treat any number of messages at the same time.

Additionally, only one object having attributes will be allowed on the left-hand side of a rule, along with the requirement that this same object must be the target of the above cited message. This restriction implements the principle of *information hiding*, which states that the internal configuration (implementation) of an object can only be visible, and therefore accessed, by itself.

Finally, although message attributes can be deleted (so they can have their value altered[1]), a corresponding attribute must be added to the rule's right-hand side, in order to prevent an object from gaining or losing attributes along the computation. Notice that this is a *rule* restriction, for if a vertex is deleted, its incident edges will also be deleted. This situation is know as deletion in unknown contexts [12], and it appears often in distributed systems, where the deletion of an object causes a number of dangling pointers to occur in the system as a whole. Rules that allow object deletion can be used to find this kind of undesirable situations within a specification.

Matches are also modified. The role of a match is to detect a situation when a rule can be applied. It occurs whenever a rule's left-hand side is present somewhere within the system graph. Notice that distinct vertices of a rule can be identified by the matching morphism. This is sensible, since an object can point to itself through one of its attributes, or pass itself as a message parameter to another object. However, it would make no sense to identify different attributes or messages, so the edge component of the matching morphism is required to be injective. Additionally, preserved elements cannot be identified with deleted elements.

A *derivation step*, or simply a *derivation*, will represent a discrete system change in time, i.e., a rule application over an actual system specified as a graph. The derivation step is also modified, to allow dynamic binding to occur.

The single-pushout approach to graph grammars extension proposed in this work (although put on an informal form) assures that the usual semantics of graph grammars can be used, and also assures that this semantics is compatible to the computation of object-oriented programs.

4 Conclusions

This work is a first step towards a very high level and intuitive formal framework compatible with the main principles of object-oriented specification and programming. More specifically it provides, in terms of object-oriented graphs and morphisms, a way of defining classes, which can be primitive or specialized from others (though the inheritance relationship) together with a graph transformation-based model of computation compatible with polymorphism and dynamic binding, which are fundamental in the object-oriented programming model of execution.

This model inherits the usual true concurrency semantics of graph grammars, making it also suitable for the specification of distributed systems. This work can be extended in many directions, amongst which there are the following:

Declarative object-oriented programming Object-oriented graphs and rules, as developed here, are in many ways similar to actual object-oriented programming language constructs. The level of abstraction used, however, is

[1] Graphs can be enriched with algebras in order to deal with sorts, values and operations. Although we do not develop these concepts here, they can easily be added to this framework.

high enough to provide a means of developing programs independent of what language will be actually used. I should be easy to translate specifications of this kind to an actual programming language. Translation from an object-oriented programming language to this sort of specification could also be interesting, since program properties could be verified if a verifier is available.

Automatic generation of (formal) specifications A number of papers describe how a grammar specification can be mapped to other formal specifications, such as Petri Nets, π-calculus, etc. Object-oriented graph grammars can also be mapped to any other formalisms, if more adequate tools for them are available.

Semantics A significant advantage to the use of a formal framework for object-oriented system specification is in the ability to apply rigorous inference rules so as to allow reasoning with the specifications and deriving conclusions on their properties. Fixing the sort of rules to be used within a graph grammar, properties regarding the computational model can be derived. Being this a formal framework, the semantics of operations (such as system and grammar composition) can also be derived.

Verification Different sort of system and grammar properties can be derived, according to the kind of rules used, the presence of values and operations from some algebra. Tools for automatic verification of properties can be constructed. Tools for type checking and static analysis can also be build, as well as abstract interpretation of programs written within this formalism.

Graph grammars are well suited for system specification, and object-oriented graph grammars, as presented in this text, fill the need for the key features of object-oriented systems be incorporated into a formal framework.

Acknowledgements
I would like to thank my thesis supervisor, Dr. Leila Ribeiro, for the valuable discussions and assistance during the course of this work.

References

1. Marc Andries, Gregor Engels, Annegret Habel, Berthold Hoffmann, Hans-Jorg Kreowski, Sabine Kuske, Detlef Plump, Andy Schurr, and Gabriele Taentzer. Graph transformation for specification and programming. Technical Report 7/96, Universitat Bremen, Bremen, 1996.
2. Dorothea Blostein, Hoda Fahmy, and Ann Grbavec. Practical use of graph rewriting. Technical Report 95-373, Queen's University, Kingston, Ontario, Canadá, Janeiro 1995.
3. Olaf Burkart and YvesMarie Quemener. Modelchecking of infinite graphs defined by graph grammars. *Electronic Notes in Theoretical Computer Science*, (6), 1997.
4. Fernando Luís Dotti and Leila Ribeiro. Specification of mobile code using graph grammars. In *Formal Methods for Open Object-Based Distributed Systems IV*. Kluwer Academic Publishers, 2000.

5. H. Ehrig and M. Löwe. Parallel and distributed derivations in the single-pushout approach. *Theoretical Computer Science*, 109:123–143, 1993.
6. H. Ehrig, M. Pfender, and H. J. Schneider. Graph grammars: an algebraic approach. In *14th Annual IEEE Symposium on Switching and Automata Theory*, pages 167–180, 1973.
7. GianLuigi Ferrari, Ugo Montanari, and Emilio Tuosto. A LTS semantics of ambients via graph synchronization with mobility. In *Proceedings of the ICTCS'01*, 2001.
8. Ana Paula Lüdtke Ferreira and Leila Ribeiro. Towards object-oriented graphs and grammars. In *6th IFIP International Conference on Formal Methods for Open Object-based Distributed Systems (FMOODS2003), to appear*, 2003.
9. J. L. Fiadeiro, C. Sernadas, T. Maibaum, and G. Saake. Proof-theoretic semantics of object-oriented specification constructs. In W. Kent, R. Meersman, and S. Khosla, editors, *Object-Oriented Databases: Analysis, Design and Construction*, pages 243–284. North-Holland, 1991.
10. Annegret Habel. *Hyperedge Replacement: Grammars and Languages*, volume 643 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1992.
11. Aline Brum Loreto, Leila Ribeiro, and Laira Vieira Toscani. Decodability and tractability of a problem in object-based graph grammars. In *17th IFIP World Computer Congress - Theoretical Computer Science*, Montreal, 2002. Kluwer.
12. Michael Löwe. *Extended Algebraic Graph Transformation.* PhD thesis, Technischen Universität Berlin, Berlin, Feb 1991.
13. Saunders MacLane. *Categories for the Working Mathematician.* Graduate Texts in Mathematics. Springer, New York, 2 edition, 1998. 314p.
14. Ugo Montanari, Marco Pistore, and Francesca Rossi. Modeling concurrent, mobile and coordinated systems via graph transformations. In H. Ehrig, H-J. Kreowski, U. Montanari, and G. Rozemberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 3 – Concurrency, Parallelism and Distribution, chapter 4. World Scientific, 2000.
15. George A. Papadopoulos. Concurrent object-oriented programming using term graph rewriting techniques. *Information and Software Technology*, (38):539–547, 1996.
16. Benjamin C. Pierce. *Basic category theory for computer scientists.* Foundations of Computing Series. MIT Press, Cambridge, MA, 1991. 101p.
17. YvesMarie Quemener and Thierry Jéron. Modelchecking of infinite kripke structures defined by simple graph grammars. *Electronic Notes in Theoretical Computer Science*, (2), 1995.
18. J. Rekers and A. Schürr. Defining and parsing visual languages with layered graph grammars. *Journal of Visual Languages and Computing*, September 1996.
19. J. Rekers and Schürr. A parsing algorithm for context-sensitive graph grammars. Technical Report 95-05, Leiden University, Leiden, Holanda, 1995.
20. Leila Ribeiro. *Parallel Composition and Unfolding Semantics of Graph Grammars.* Phd thesis, Technische Universität Berlin, Berlin, June 1996. 202p.
21. Andy Schurr. Programmed graph replacement systems. In H. Ehrig, H-J. Kreowski, U. Montanari, and G. Rozemberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 1 – Foundations, chapter 4, pages 479–546. World Scientific, Singapore, 1997.
22. Gabriele Taentzer and Hartmut Ehrig. Semantics of distributed system specifications based on graph transformation. Lecture Notes in Computer Science.

23. Gabriele Taentzer, Ingrid Fischer, Manuel Koch, and Victor Volle. Visual design of distributed systems by graph transformation. In *Handbook of Graph Grammars and Computing by Graph Transformations*, volume 3 – Concurrency, Parallelism, and Distribution, chapter 1. World Scientific, 2000.

Assembling contracts for components

F. Legond-Aubry, D. Enselme, G. Florin

Conservatoire National des Arts et Métiers,
292 rue Saint Martin,
75005 Paris
`fabrice.legond-aubry@lip6.fr, {florin,enselme}@cnam.fr`

Abstract. Components are increasingly used to create complex and distributed systems and applications. They are often viewed as simple servers, which limits their capacity at collective action. In this article, we propose a method to simplify their assembly and their potential re-usability. We use the notion of dependency and contract between components to explicitly design an entity that guarantees the correctness of the built system. We introduce split contracts and delegations of properties to check, both at conception and execution time, the correctness of the assembly. An added value is that our solution increases the independence of the participating entities by isolating the core components and transferring the aggregation into specific "glue" components.

Keywords : Contract, Assembly, Component, Constraint, Interaction, Dependency, Specification

1 Introduction

The Object Oriented approach was introduced to offer powerful tools and efficient structural design. However complexity remains in the interactions that tightly bind each object with the others. This hinders the development of large scale industrial applications. Components were introduced to enhance the isolation and the separation by increasing the granularity of the manipulated entities, and by giving them new capabilities. But if the encapsulation gives abstraction power, it hides the specification of the component (and mainly its internal behavior). Components are too often black boxes without a "user manual".
Contracts were introduced in the objects paradigm by Helm [5]. This aimed to compensating the lack of methods to express the relations between objects. They were used to specify behavioral compositions. Using contracts provides an orthogonal dimension to the one provided by the class structure. Techniques of development are more and more based on the component approach [4]. To improve these techniques and allow re-use, contracts were extended and adapted to them by Meyer [10] and Jezequel [6]. Generally, contracts exist for server only or for both the client and the server. The first proposition induce a significant anti-symmetry of the call as they do not care about clients constraints and the second one melt all the constraints about the two entities without separating their respectives concerns.
This article takes place in an industrial research project called ACCORD[1]. Its main objective is to propose to applications creators an integrated development environment with analysis tools that use more symmetric contracts for describing component assemblies. This paper begins by stating current trends in the utilization of the contracts for components. Then we introduce a new notion of the call for a component service. From this point, we set up contracts for this type of call and define a notion of compatibility between two contracts. With this basic definition, we then introduce an extended compatibility notion to n components. To illustrate our approach, we present a simple cash dispenser example.

[1] http://www.infres.enst.fr/projets/accord/

2 Classic component interaction point of view

A component is a cooperative composite entity similar to those described in the Architecture Description Languages [8]. A component has interfaces which are sets of operations - also called methods. Sometimes interfaces are grouped to make ports which become a point of interaction. A first attempt to introduce contracts in components was done in 1999 in the article "Making Component Contract Aware" [3]. It introduces four types of contract : syntactic, behavioral, synchronization, and quality of service. It uses conditions and a unique contract carrying both client and server constraints.
An entity that interacts can only accept the contract (or one of its siblings) and respect it. This point of view limits the assembling capacity of a developper because some entities could have different specifications that are compatible without having any obvious relationship - ie same contract. Another restriction is that the contracts are to be used during execution to verify the validity of an interaction. Finally, they are dedicated to an application even if adaptation remains possible. Constracts are made for a specific context therefore it is difficult to extract a component from the whole.
Tools like Jcontract [14], Icontract[16], Eiffel[9] implement "Design By Contract" assertions but for an Object Oriented Model not for components. Moreover they are use for test purpose and not at all for model checking.

3 Toward a more symmetric notion of the invocation

3.1 Definition

To relieve these "dependencies" and "checking" limitations, we choose to introduce "split" contracts. Each element has its own requirements and its own guarantees expressed as a set of properties. If two entities have to interact, the sum of their split parties set a contract for the interaction. We are currently out of a specific platform model like COM+, EJB, CCM, .NET [11,2,17]. All notions will be kept abstract as far as possible. Though, in this abstract model, stress is put on the semantic and the pragmatic viewpoints that is to say on the functional and non-functional properties of a component. This document mainly deals with the description of interactions between components using contracts but wholly viewed as collaboration [13].
In fact, the contracts define the relations of dependence and interactions involving the elements of the application. Following this idea, we associate, as in the CORBA CCM Model [17], at each interaction the specifications which accurately set the required service context (the client point of view) and the offered service context (the server point of view). These specifications are expressed by pre and post conditions. These assertions determine the guarantees and the obligations applying to each participant of the interaction. That is to say that even the client can specify required properties that the server must provide. If the server does not provide them, the client/server binding won't be established.
To define the compatibility between two operations, we extend the classical sub-typing notion: a type T is a subtype of a type T' (noted $T <: T'$) if all values of T can be used in a consistent manner for each expected values of T'. We denote $PO <: RO$ the compatibility of the specifications of a Required Operation (RO) with the specifications of an existing Provided Operation (PO). Generally the service are under-specified but this is the only data (aside testing) that we can rely on. Moreover, programmers always use the specifications for the important parts of the new designed component. So when we talk about compatibility between PO and RO, we mean compatibility between their respective specification: $S_{PO} <: S_{RO}$. To make these specifications cover all fields of the development phase, they must describe and be verifiable at a syntactic, semantic and pragmatic level.

3.2 The syntactic level of compatibility

At the syntactic level, the compatibility relationship uses subtyping. We enforce that an operation o of type T is syntactically compatible with an o' operation of T', if they have the same number

of parameters and the same identifier, if the contravariance of in parameters and the covariance of out parameters are confirmed. The contravariance of in parameters asserts that the parameters are in reverse relation of subtyping accordingly to the notion. The covariance of the out parameters asserts that the parameters carry out the relation of subtyping accordingly to the notion.
Eg :

- `[Provided] Operation : void an_operation (in long parameter)`
- `[Required] Operation : void an_operation (in int parameter).`

Eg :

- `[Provided] Operation : void an_operation (out int return_param)`
- `[Required] Operation : void an_operation (out long return_param).`

This prohibits the subtyping relationship for the in/out parameters. This definition is one possibility among many others but it has the advantage of limiting the semantic variations of operations having the same signature (syntactically compatible) but with a totally different semantic. There is no automatic solution. Only the human brain can make the difference. As this relation of syntactic compatibility relies on subtyping notions, the relation is transitive but is not symmetric so we have $\neg(PO <: RO \Rightarrow RO <: PO)$.

3.3 The semantic level of compatibility

In component interaction, the specification S_{RO} of a Required Operation (RO) is a set of properties (that can be expressed as an OCL constraint by sample) required to prove the correctness of the client. In a similar manner, a provided operation (PO) and its specification S_{PO} guarantees the “usage” properties of all correct implementations of a server. Figure 1 ilustrates this issue.

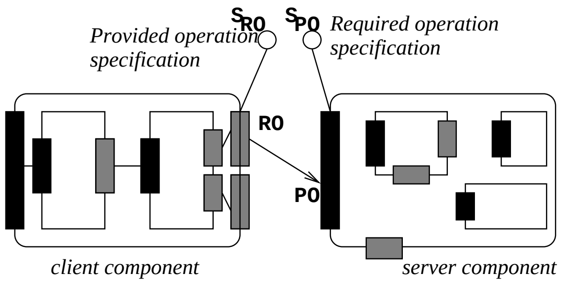
S_RO
S_PO
Provided operation specification
Required operation specification
RO
PO
client component
server component

Fig. 1. Interaction between a provided and a required service

A call correctness can be asserted by determining the conditions in which a client wishing to use a service RO could use a PO service instead. The correctness of a call is stated as conditions to satisfy for RO to be replaced by PO. We choose to use the formalism of pre and post conditions. In a non-distributed programming context, assertions are expressed in first order logic. Nevertheless, it is generally widely recognized that such a logic is not adapted to concurrent and distributed environments. The behaviors encountered in such environments can not be easily modelized with this simple logic. Hence, whenever concurrence and distribution come into play, higher level logics are generally chosen. For instead, modal logics with temporal (for concurrence) and epistemic (for distribution) [15] operators, are some of the most widely encountered solutions. Like with first order logic, pre and post predicates are used. This pre predicate (O_pre) is evaluated just before the execution of the operation and sets the conditions of the proper realization of the interaction (and subsequently the service). But, in most cases, the pre and post conditions bind a set of constraints that does not specify one unique coherent state but a family of possible acceptable

execution historic that will lead to a valid execution. In the same way, a post condition (O_post) defines a set of potential correct futures (after the achievement of the service).
Leaving apart the case of distribution, we give a clue on the way temporal logics comes into play. Many such logics have been defined in the literature, the most significant one being maybe Lamport's Temporal Logic of Action [7]. In TLA, an execution is viewed as a sequence of steps, each producing a new state by changing the values of one or more variables. We will consider an execution to be the resulting sequence of a succession of states that will take the semantic meaning of the studied component interactions. We apply the same analogy here. At the very instant where the operation is enabled then the O predicate is true. "O" denotes the state where the code of the operation O is ready to be executed. The same holds for the $O \wedge O\_pre$ (pre-conditions are verified). After the execution of the operation O, the operation following O (noted $nextO$ or XO) is enabled. The predicate $XO \wedge O\_post$ is true at this very moment. This invocation specification presumes the atomicity of the invoked operation. At this granularity level, we do not provide any informations on the behavior of external invocated O operation during its execution as soon as it respects the invoker (client) requierement. So basically the execution can be symbolized by $O \wedge pre \Rightarrow XO \wedge O\_post$ as in figure 2.

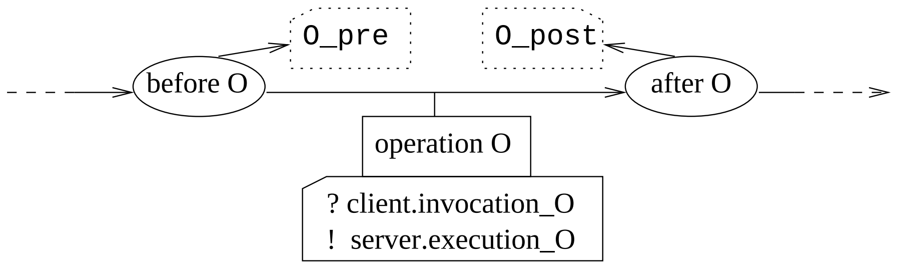

Fig. 2. Temporal Logic Representation of an operation execution

In this context, we must introduce time and causality and epistemic expression tools and languages but this is not the purpose of the current paper. Epistemic deals with knowledge that the component must or must not have a component about the others and how it can learn it. The previous basic statement can be extended to express the semantic and temporal compatibility of an interaction by using the five following equations:

1. $RO \wedge RO\_pre \Rightarrow XO \wedge RO\_post$
2. $PO \wedge PO\_pre \Rightarrow XO \wedge PO\_post$
3. $RO = PO \wedge XRO = XPO$
4. $RO \wedge RO\_pre \models PO \wedge PO\_pre$
5. $XPO \wedge PO\_post \models XRO \wedge RO\_post$

The first condition (1) tells that there exists a specification S_{RO} of a RO which enables the client to move from one coherent state to another. The second expression (2) states the existence of a provided operation specification (S_{PO}) which enables the server to pass from one coherent state to another. The last three conditions seal the relation of compatibility between the client and the server (noted by the $\models$ relation). The third condition guarantees that the provided operation PO can be used instead of the required operation (RO) or in other words that the component can make a direct invocation without the need of a connector or an adaptator to link the PO and the RO. The fourth condition imposes the contravariance between the provided and required pre-conditions. The fifth condition imposes the covariance between the provided and required post-conditions. In other words, before the invocation, each pre-condition from the callee must be verified by the caller and at the return, each post-condition from the caller must be in agreement with the properties of the callee.

3.4 The pragmatic level of compatibility

The pragmatic compatibility is the most difficult to define. It consists in the behavioral and environmental compatibility verification. Pragmatic properties are often difficult to analyse. They could be expressed totally by logical expressions but also as state-transition diagram (ie stat-chart diagrams). In this case specifying compatibility is equivalent to testing if the diagram of the PO is less constrained than the diagram of the RO. Typically, this is the notion of symmetries found in Petri nets. This type of compatibility will be transitive but the symmetry will not be guaranted as it was in the other levels. A more complex definition of the compatibility can be defined for interfaces, which will impose a causal order between calls. However the tools and technics are exactly the same than those used to compare methods behavior.
Semantic and pragmatic constraints can be expressed with first order predicate logic[2], second order modal logic[3] [1] and with different languages. Our study does not limit itself to one of them but all services are to be specified using the same logic language. It is not our purpose to develop our own language so we check some of them among the abundance of existing ones. One of the most widely used is OCL. In addition, there are temporal extensions of it and some software checkers are available for free [12]. One thing is yet missing in OCL : it is a epistemic (knowledge) expression extension. This add-on could be very useful to formulate and manage the ways and means of the diffusion of the informations and knowledge of a specific data among all the component entities. A possibility to deal with theses properties is to use a runtime checker, to test if a component bind itself with an unauthorized entity or/and get access to forbidden data.

3.5 Interaction contracts

One of the essential principles in an interaction contract elaboration is the compatibility between the properties of each contributor. The sufficient and necessary condition to the existence of this contract is the compatibility between the two involved operations.
So we must have $S_{RO} <: S_{OO}$. If this hypothesis is not verified then the interaction and subsequently the assembly is not possible. An interaction contract has a specification that comes from a negotiation process between the client and the server on the base of the required and offered specifications. So a simple interaction contract is a specification S_{CO} which uniquely typify an interaction. The specification of the contract replaces, in the application using the caller and the callee, their respective specifications (S_{PO} and S_{RO}). The existence of the S_{CO} is ensured by the transitivity of the relation so that we have:

- $PO <: CO <: RO$
- $RO\_pre \models CO\_pre \models PO\_pre$ `(contravariance)`
- $RO\_post \models CO\_post \models PO\_post$ `(covariance)`

The $\models$ states the properties compatibility.

3.6 More than interactions contracts : delegation of properties

We observe some specific cases in the construction of the specifications of S_{RO} or S_{PO}. On one hand the programmer can use a « Component Off The Shell ». In this case, s/he can take the provided specifications defined by the "COTS". Henceforth, the developer accepts the post conditions *PO_post* and makes them his own *RO_post* for the new component that must be implemented. In the same manner, he uses the preconditions *PO_pre* for the *RO_pre* and tries to satisfy them in the code of the components. Therefore, the newly created component accepts all the imposed conditions by adopting the specifications of the component which it is foreseen to interact with.

[2] When the subject of the sentence is an individual object (like Socrates in "Socrates is mortal"), then we are using first order logic
[3] When the subject is another predicate (like being mortal in "Being mortal is tragic"), then we are using second order logic or higher order logic

Verifying the peer to peer compatibility between the two elements is then trivial. This case often rises in down-top development methodologies.
On the other hand, when the programmer develops a new component, s/he can use the specifications of a required operation to build a new component with a provided operation that will totally satisfy the required properties. This is the case of bottom-up methodologies.
But this is insufficient. A composite is composed of a set of components accordingly to an assembly graph. As a component can be a composite itself, the abstract model is fractal. If there is no dependence between the specification then the problem of constructing the assembly is reduced to a succession of definitions of interaction contracts. But in the general case, there are dependencies between the specifications so we must verify the validity of the aggregation.
Dependencies are reified by the delegations of properties between point of interaction of the components. Their are different types of delegation. The first is the pure delegation. In this case, the provided specifications of an externally visible operation S_{EPO} is the same as the specifications of an internally provided service (S_{IPO}) of a member of the composite. Another working hypothesis would be a delegation by compatibility. Then a specification of a externally visible provided service is compatible with the provided service of a member of a composite ($S_{EPO} <: S_{IPO}$). Another last solution would be a delegation by adaptation so that the provided externally visible service would be an adaptation of an internal provided one.

4 A trivial illustrative example

Firstly, let us say that more informations and a more complex sample can be found on the project page of ACCORD [4]. But, to illustrate this, let us take a naive getCash operation, which withdraws cash from a bank account. The client coordinator has a required interface RIBank link with the bank component obtained through a provided interface PIBank from a bank component. The specified method is getCash. The Required Operation of the client component is:

```
Component Coordinator - interface RIBank
  Operation getCash (amount, account)
    precondition (OR_pre) :
      type (amount) is type_amount
      ∧ type (account) is type_account
      ∧ amount < maximum1
      ∧ balance (account) amount> 0
    postcondition (OR_post) :
      Bank.PIBank.getCash.post
```

To be more precise, it would have been interesting to introduce the epistemic modalities where the client would only know if the operation is possible or not without knowing the balance. However, the previous code only limits the amount of money a client can take to a maximum of *maximum1* and imposes that the balance of the account will be positive after the operation. As a client, the cash dispenser accepts all post-conditions imposed by the bank component it is in relation with.
The specification S_{PO} of the provided operation (PO) is:

```
Component Bank - interface PIBank
  Provided Operation : getCash (amount, account)
    precondition (PO_pre) :
      type (amount) is type_amount
      ∧ type (account) is type_account
      ∧ amount < maximum2
      ∧ balance (account) amount > 0
    post-condition (PO_post) :
      balance (account)=balance (account)-amount
      ∧ balance (account) < triggerLevel(account) ==> msg_alert
```

This specification only enables *getCash* operation with an amount which is less than *maximum2*. The post condition throws an alert message if the amount on the account is below a trigger value.

[4] http://www.infres.enst.fr/projets/accord/lot1/index.html

In this case, the contravariance hypothesis on the preconditions ($RO\_pre \Rightarrow PO\_pre$) enforces the following condition for the operation to be successful:

```
    amount<maximum1 ==> amount<maximum2
```

The maximum amount required must be less than the maximum amount provided. The covariance hypothesis on the postconditions ($PO\_post \Rightarrow RO\_post$) are naturally satisfied because the invoker accepts all the conditions of the server:

```
    [ balance (account)=balance (account)-amount
    ∧ balance (account) < triggerLevel(account) ==> msg_alert ]
  ⟹ balance (account) = balance (account) amount
```

So from these conditions, we can make a S_{CO} which is compatible with S_{RO} and S_{PO} for the interaction between RO and PO :

```
Contract Operation: getCash (amount, account)
  precondition (CO_pre) :
    type (amount) is type_amount
    ∧ type (account) is type_account
    ∧ amount < maximumC
    ∧ maximumC < maximum2
    ∧ balance (account) amount > 0
  postcondition (CO_post) :
      [ balance (account)=balance (account)-amount
      ∧ balance (account) < triggerLevel(account) ==> msg_alert ]
    ==> balance (account) = balance (account) amount
```

In S_{CO}, we must adopt in the pre-condition with an adapted maximum called *maximumC* which must be less than *maximum2* and in the post-conditions those from the client which are compatible but which enforce some more conditions.

5 Future work and Conclusion

In this article, we present a new method for assembling components and some tracks to reduce the time for their assembly. This method can be applied in an ascending manner (during conception) or in a descending manner when we want to accurately design an aggregation of existing components. The expression of split contracts allows all handlings to be done with unique contracts. It also enables to have a better separation of each element, a new easier way to gather them and a possibility to do some structural verification of the developed application by the mean of a method based on a fixed point equation resolution. Actually, we hope to implement this method to at least localize interfering components and assembly anomalities subsequently easing the rectification of the assembly by the conceptor. Then, we plan to implement a projection of them on multiple component platform using aspects so that simple modifications could be more easily implemented in a shorter time and to introduce .

The second part, is the code "patcher" which enables programmers to implement their transformations contracts directly in the deployed component application. Aspect Oriented Programming (AOP) enables us to easily set orthogonal intervention points (also known as pointcuts) in the framework and the components code. AOP fits very well to the need of assembling and composing components because we can set these complex "intervention" points everywhere in the component. Java is the most efficient language because it has the particularity of being OS and platform independent. Moreover many components platforms are already implemented in Java.

Type of call	Duration for 60000 calls (in ms)
Standard local call	Not measurable
RMI call	27480
Call of a JAC "patched" method (without any aspect)	61
Call of a JAC "patched" method (with two dumb aspects)	110

Fig. 3. Overload induced by the JAC platform on a method call

Dynamic AOP is the most efficient tools because it enables dynamic modification and reaction to the environment. Previous studies conducted with the JAC platform taught us that the overload induced by the cost of dynamic wrapping is negligible compared to the latency induced by remote calls. Table 3 illustrates this issue.

References

1. Second order logic, foundations, and rules. 1990.
2. A.Thomas. Enterprise javabeans technology. White paper, Sun Microsystems, Inc., 1999.
3. Antoine Beugnard, Jean-Marc Jézéquel, Noël Plouzeau, and Damien Watkins. Making components contract aware. In *IEEE Software*, pages 38–45, june 1999.
4. D. F. D'Souza and A. C. Wills. *Objects, Components and Frameworks with UML: The Catalysis Approach*. Addison-Wesley, http://www.catalysis.org, 1998.
5. R. Helm, I. M. Holland, and D. Gangopadhyay. Contracts: Specifying Behavioral Compositions in Object-Oriented Systems. In *Proc. of the OOPSLA/ECOOP-90: Conference on Object-Oriented Programming: Systems*, pages 169–180, Languages, and Applications / European Conference on Object-Oriented Programming, Ottawa, Canada , 1990.
6. J.-M. Jézéquel, M. Train, and C. Mingins. *Design Patterns and Contracts*. Addison-Wesley, October 1999.
7. Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(3):872–923, 1994.
8. Nenad Medvidovic and Richard N. Taylor. A classification and comparison framework for software architecture description languages. *Software Engineering*, 26(1):70–93, 2000.
9. B. Meyer. *Eiffel : The language*. Prentice-Hall, 1991.
10. B. Meyer. Applying Design by Contract. *Computer*, October 1992.
11. Microsoft. .NET. *http://www.microsoft.com/net/default.asp*, 2001.
12. Klasse Objecten. OCL Checker. Web site, Klasse Objecten, 1999.
13. P. Champagnoux, L. Duchien, D. Enselme, G. Florin. Typage pour des composants coopératifs. In *NOTERE 2000 Proceedings*, November 2000.
14. Parasoft. jContract. Web site, Parasoft, 1996.
15. R. Fagin,J. Y.Halpern, Y. Moses, M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, 1995.
16. R. Kramer. iContract - The Java Design by Contract Tool. Web site, iContract, 1999.
17. R.Marvie. Corba components: la proposition unifiée, du modèle au objet au modèle des composants. Technical report, LIFL, May 1999.

Enabling Re-configurability on Component-based Programmable Nodes

Jó Ueyama, Stefan Schmid, Geoff Coulson, Gordon S. Blair, Antônio T. Gomes, Ackbar Joolia, Kevin Lee

Computing Department
Lancaster University
LA1 4YR Lancaster, UK
{ueyama, sschmid, geoff, gordon, gomes, joolia, leek}@comp.lancs.ac.uk

Abstract

Recently developed networked services have been demanding architectures that accommodate an increasingly diverse range of applications requirements (e.g. mobility, multicast, QoS), as well as system requirements (e.g. exploiting specialized processing hardware). This is particularly crucial for architectures of network systems where the lack of extensibility and interoperability has been a constant struggle, hindering the provision of novel services. It is also clear that to achieve such flexibility these systems must support extensibility and re-configurability of the base functionality subsequent to the initial deployment. Based on our experience with middleware platforms, we argue that re-configurability of network services is best implemented by means of reflection.
In this position paper we present component-based approach to developing flexible networked systems, especially targeted at the Intel IXP1200 programmable networking environment and discuss how our approach can offer a more deployable, flexible and extensible networking infrastructure. We show the viability of our component model to re-configure services on the Intel IXP1200 platform. An application scenario is presented to validate our approach to enable re-configuration of services across different layers of an IXP1200-based router platform.

1 Introduction

An increasing number of recent applications (e.g. real-time, multimedia) and their underlying system infrastructures (e.g. workstations, PDAs, embedded systems, ad-hoc networks) have been requiring a flexible architecture to accommodate all the requirements necessary to run these applications as well as to inter-operate in a heterogeneous environment composed of different types of applications and hardware platforms. It is increasingly clear that to achieve this, we need an extensible and re-configurable architecture that is capable of loading and integrating new functionality at run-time. As an example of re-configurability, we could load and unload services on a network router and intelligently adapt its forwarding behaviour to various types of traffic and environments such as mobile or ad-hoc.

Unfortunately, although much research on providing an open architecture for networking systems have been carried out, we still lack a generic approach to develop and deploy new network services. Existing paradigms address configuration and re-configuration of services running on a particular level of a programmable networking system (e.g. open signalling for *control functions*, and active networks for *in-band packet processing*). But there is as yet no really comprehensive approach.

At the same time, component technology [21] has been widely cited as a suitable model for developing adaptive software due to its incrementally deployable nature [10]. For example, with component technology described in [10] one can add, replace and remove the constituent components residing in the same address space. Therefore, the use of component technology provides a means for deployment-time configurability and run-time re-configurability. However, although component-based architecture have been successfully used in many adaptive applications, early research into active networks has not truly adopted a component model [18]. Moreover, the majority of existing work (e.g. Vera [15] and Genesis [5]) omit support for dynamic re-configuration. RANN (*Reflective Active Network Node*) [22] introduces the use of reflection to support flexible configuration in active networks, but it only defines an architecture where active nodes use reflection to better structure services. Essentially, RANN defines an architecture for configuration rather than re-configuration. Moreover, we argue that this work is partial; it only addresses the configurability within the execution environment, and not at lower or higher system levels. Furthermore, RANN is language specific (Java).

As a consequence, this paper presents the design and implementation of a component-based architecture for programmable networking software, which provides an integrated means of developing, deploying and managing such systems. The proposed architecture consists of a generic component model applied on *all levels* of the programmable network design space, which ranges from fine-grained, low-level, in-band packet processing functions to high-level signalling and coordination functions. The projected benefits of this approach are detailed in section 4. Configuration and re-configuration across this architecture is achieved by dynamically loading and unloading service components. Reflection is used to reify configurations of components and to support various types of meta-data to facilitate configuration and re-configuration.

The remainder of the paper is structured as follows. Section 2 looks in detail at existing component-based frameworks for generic applications and component-based technology to provide programmability in networks which leads to the motivations of our work. Subsequently, section 3 examines existing technology necessary for this architecture. It introduces our component model OpenCOM, the concept of component frameworks and finally covers the IXP1200 platform. Section 4 then reports on our globally applied component-based architecture to enable dynamic creation, deployment and management of services in programmable networking environments. Our approach and an overview of the design space of the programmable networking environment is detailed in the same section. The design and implementation of our model carried out to date is presented in section 5. Finally, section 6 draws general conclusions from this paper and the proposed architecture.

2 Analysis of Existing Work

We distinguish here component-based platforms for general purpose applications from component models tailored towards a specific application domain (such as active and programmable networks).

2.1 Generic Component-based Platforms

Although there exists a wide range of research systems (e.g. XPCOM, K-Component [11]) that implement component platforms for generic applications, we argue that all of these are suboptimal for programmable networks due to their lack of special support for the hardware platforms used in this context. These limitations are frequently related to the platform upon which the component is built for (e.g. the Java virtual machine). Another limitation regards the system layer at which the component model is targeted. For example, none of the existing component models consider the integration of low-level and high-level components running on different layers of the router hardware (e.g. in-band packet forwarding, or signaling). There exists work addressing low-level components providing an architecture to build component-based OSs (e.g. Think [12], Knit [17]). Nevertheless, typically these systems do not provide a uniform and globally applied framework to load and bind both assembly language-based components and high-level components.

2.2 Component-based Platforms for Programmable Networks

Aside from the component model for generic applications, there has been considerable research on component-based platforms for programmable routers (e.g. Click [16], NP-Click [20], VERA [15], NetBind [5], LARA++ [19]). Nevertheless few works support configuration and reconfiguration (i.e., adaptation, extension, evolution and removal) sufficiently. Most of them support the first configuration but do not support the subsequent re-configuration at runtime. Moreover, systems that implement re-configuration do not adequately support the management of system integrity over re-configuration operations (e.g. ensuring that firewall updates are applied consistently and universally). Furthermore, these works do not provide an *integrated* approach to configure and re-configure services across all layers of the programmable networking system (see section 4). For example, VERA limits re-configurability to in-band functions and the hardware abstractions layer, whereas NetBind considers only in-band functions. LARA++, on the other hand, allows re-configurability on all layers, but lacks an uniform model to do so (i.e. different component models are used on the different layers).

3 Background

3.1 Component Frameworks

The concept of component frameworks (CFs) is applied to deal with component constraints and the dimensioning of the applicability of participating components. CFs are also applied to provide the *structure* allowing the use of components for a specific domain of application. A number of runtime CFs have been implemented as part of our past research (e.g. pluggable protocols, pluggable thread schedulers, and pluggable media filtering) [10]. CF were originally defined by [21] as *collections of rules and interfaces that govern the interaction of a set of components "plugged into" them.* In this sense, a CF embodies rules and interfaces that would make sense for a specific domain of application. For example, a CF for pluggable protocols consists of a variety of interfaces and rules to integrate plug-in protocols and ensure that they are stacked in an appropriate order. As another example, CFs can determine constraints to govern the interaction of a set of components: a CF can mandate that a packet scheduler component must always read its input from a packet classifier. Such constraints are useful to ensure meaningful re-configuration and therefore the system must provide support for expressing these constraints. Essentially, CFs provide the necessary support and conditions for components and also regulates the interaction rules between component instances for a specific domain of application. A component framework can come alone or interact and cooperate with other CFs (as long as it conforms to the rules governed on the host CF). Therefore, it is natural to design CFs themselves as components.

3.2 OpenCOM

Lancaster's OpenCOM [7] is a lightweight, efficient, flexible, and language-independent component model that was originally developed as part of previous research on configurable middleware [10]. OpenCOM is fine-grained in that its scope is intra-capsule (for capsule, see below) and it imposes a minimal overhead on cross-component invocation without compromising the overall performance. It is currently implemented on top of a subset of Mozilla's XPCOM platform.

OpenCOM relies on four fundamental concepts:

Capsule: a capsule (see figure 1) comprehends multiple physical address spaces within a single logical container: for example, a capsule could encapsulate both a Linux process in the IXP1200's control processor and one or more microengines. Encapsulating multi-address-spaces offers a powerful means of abstracting over heterogeneous but tightly-coupled hardware (e.g. PC, StrongARM and the microengines of the IXP1200 router platform - see section 3.3 and figure 3 for the IXP1200 architecture).

Interface: expresses a unit of service provision;

Receptacle: define a service requirement and is used to make the dependency of one component on another explicit;

Binding: is an association between a receptacle and an interface. In the original version of OpenCOM, bindings were restricted to receptacles and interfaces residing in the same address space. Currently, OpenCOM is being extended to support binding between receptacles/interfaces of different address spaces as well. Semantically, a connection represents a communication path between one receptacle and one interface. Bindings in the original OpenCOM were exclusively implemented in terms of vtables [3] (a vtable is essentially a table containing pointers to virtual functions). Currently, however, we are extending OpenCOM to support bindings implemented in a variety of ways.

On top of OpenCOM, we have designed a CF called the "Router CF" that is targeted specifically at packet-forwarding functions in routers. See figure 2. In this illustration the *meta-interfaces* support

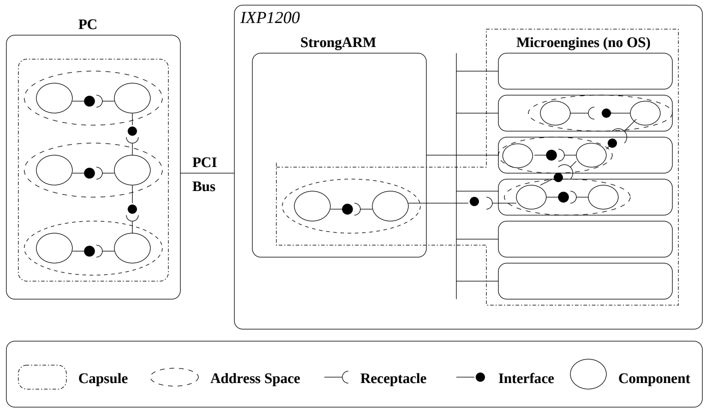

Figure 1: Multi-address-space capsules

inspection of the types of interfaces and receptacles declared by the target component. The meta-interfaces use the underlying XPCOM system to query the component's type library file and return the IID's (interface identifiers) of the interfaces it implements. Information obtained from this inspection can then be used to enable and guide subsequent re-configuration (e.g. to replace one component with another). Reflection is used as the means to obtain such information, which enables the re-configuration in our framework. Moreover, the obtained information is useful in enabling CFs to check the type of interfaces offered, enforcing the compliance of binding rules at runtime. The *controller* component (see figure 2) relies on the mentioned meta-interfaces in order to manage and configure the internal constituents of the respective CF. It is important to emphasize that the CF can be composed hierarchically of other CFs of the same type, in order to build a composite of CFs.

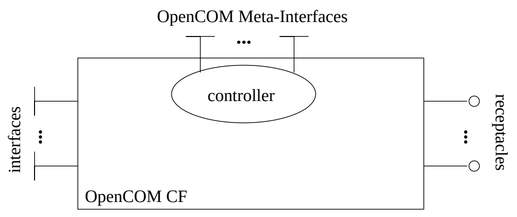

Figure 2: Generic CF for programmable routers

3.3 The Intel IXP1200 Router

The IXP1200 router [8, 13] is an Intel-proprietary architecture based on IXP1200 network processor (NPs). Its architecture combines a StrongARM processor with six independent 32-bit RISC processors called *microengines*, which have hardware multithread support. The StrongARM is the core processor and is primarily concerned with control and management plane operations, whereas microengines handle packet-forwarding. Our component model is being adapted for this platform. In terms of memory, the IXP1200 platform provides the *Scratch* or *Scratchpad* and also the static and dynamic RAM. Figure 3 illustrates the implementation environment of our prototype.

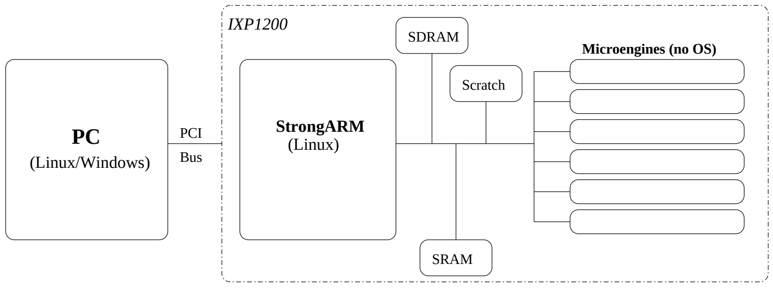

Figure 3: Simulation environment of our prototype

4 Our Approach

4.1 Characterizing Programmable Networking Environments

The design space of programmable networking [9] can be split into four layers or “strata”. We use the term “stratum” rather than “layer” to avoid confusion with layered protocol architectures. The four strata (see figure 4) are described as follows:

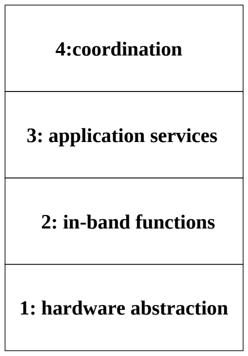

Figure 4: Design Stratas of Programmable Networking

Hardware Abstraction: The *hardware abstraction* stratum corresponds to the minimal operating system functionality needed to run applications on the higher levels. This functionality encompasses threads, memory, I/O, library loading and other services needed to support higher-level network programmability. Interfaces in this stratum are often implemented as *wrappers* around underlying native facilities in order to support heterogeneous platforms which allows transparent use of these services on a number of different router architectures, such as PC-based router or Intel IXP1200s which provides multiple processors (StrongARM and microengines) and distributed/hierarchical memory arrays.

In-Band Functions: This stratum consists of packet processing functions such as packet filters, checksum validators, classifiers, diffserv schedulers, and traffic shapers. Given that these functions are low-level, in-band and fine-grained (and therefore highly performance critical) they must be implemented extremely efficiently (i.e. machine instructions should be counted with care.

Application Services: The application services stratum encompasses coarser grained functions: for example, active networking execution environments [1]. Functions in this stratum are less performance critical and act on pre-selected packet flows in application specific ways (e.g. per-flow media filters).

Coordination: This stratum supports out-of-band signalling protocols which carry out distributed coordination, including configuration and re-configuration of the lower strata. This, for example, includes signalling protocols such as RSVP, or architectures that enable resource allocation in dynamic private virtual networks as employed by architectures like Genesis [4], Draco [14], or Darwin [6].

The main aim of our work is to provide a globally-applied component model, which can enable (re)configuration of services in *all* strata of the programmable networking design space. This yields a number of important benefits. The component model:

- *is simple and uniform* - it allows the creation of services in all strata and provides a uniform run-time support for deployment, inspection and (re)configuration;
- *enables bespoke software configurations* - by the composition of CFs in each stratum, desired functionality can be achieved while minimising memory footprint; trade-offs vary for different systems types (e.g. embedded, wireless devices; large-scale core routers);
- *facilitates ad-hoc interaction*–e.g. application or transport layer components can directly access (subject to access policies) "layer-violating" information from the link layer, which nowadays is considered as indispensable [2].

We aim to apply this approach in both PC-based router as well as specialised programmable routers (e.g. Intel IXP1200). This heterogeneity is fundamental to validate our claim of a generic model. We also strive to implement this model without compromising the overall performance.

4.2 Aims and Objectives

Our project aims to build a framework to enable re-configuration of services running on all levels (*strata*) of any programmable router. It attempts to mask underlying hardware heterogeneity, so that a PC-based router and an IXP1200-based router which has multiple processors (i.e. *microengines* and *StrongARM*) and distributed/hierarchical memory array, will look as similar as possible to other strata. Reconfigurability in this architecture will be carried out by means of reflection technology which is implemented in OpenCOM.

Initially, we are building a version of the Router CF described above for the IXP1200. Essentially, this CF offers an abstracted view of the IXP (i.e. an API) to both developers and deployers of programmable networking functionality. The CF embodies rules and constraints that component developers must follow in developing plug-in components.

5 Progress to Date

In line with the above-mentioned aims and objectives, our component model for programmable networking platforms enables *loading* and *binding* of components residing in all strata of the design space of the programmable networking environment. The core architecture contains essentially these two functionalities: *loading* and *binding*. The former loads a class (i.e. code) and also creates an instance of one component in the specified *capsule_id* and the latter accomplishes the binding between individual components (between interfaces and receptacles).

5.1 Generic Framework

Following the above-mentioned approaches and aims, we extended the core API of OpenCOM with the following methods:

- `load(capsule_id, component_guid);`
- `unload(capsule_id, component_guid);`
- `bind(interface_guid, receptacle_guid);`

- `unbind(interface_guid, receptacle_guid);`

In these methods, *capsule_id* specifies either the capsule where the component will be loaded or the capsule where the component resides (e.g. in the PC, StrongARM or a Microengine). *Component_guid*, *interface_guid* and *receptacle_guid* are component, interface and receptacle identifiers respectively. In our prototype, GUID is the "*globally unique identifier*" and it is used to locate a specific component, receptacle, and interface to be loaded or bound. For high-level components, we employ the ClassID (implemented in XPCOM) as our identifiers for components; and *InterfaceID* as interface and receptacle identifiers. For Microcode-based components, we have been using the name and the path where each component is located to identify the component and its interface and receptacle.

An example environment is depicted in figure 3. Note that this example employs three address spaces: PC, StrongARM and Microengines.

5.2 Re-Configuration: Loading and Binding using Reflective Techniques

The loading mechanism determines the physical placement of components to-be-deployed by taking into account factors such as resource usage as well as QoS and security constraints. Similarly, the binding mechanism wires the component into the appropriate place in the software router configuration. The implementation of the above functionality includes a number of wrapper functions that map each *load* and *bind* method to the corresponding mechanism to re-configure components in the requested address space.

Components running in this environment can typically be of any granularity, i.e., components can be coarse-grained (e.g. developed for the StrongARM or PC platform) or fine-grained, which in our case are typically coded in the Intel IXP1200 machine language (i.e. microcode). This enables us to write high-level services that rely on functionalities provided by low-level components.

In this sense, the binding mechanism can as well connect components of any type. Bindings between microengine and StrongARM components is (typically) carried out by using a shared-memory mechanism (exploiting the IXP1200s SDRAM and SRAM common memory). We have also been using the *Scratch memory* (for Scratch memory, see figure 3) to transfer integer values between these components (see figure 5 for details) as this provides the lowest transfer time (compared to SRAM and SDRAM). Finally, binding between two microengine components is achieved by changing the execution path from one component to another, i.e., by changing the branch instruction, pointing the execution path to the next component (see figure 6). This is implemented in the same way as in NetBind [5]. However, we claim that our implementation is more fine-grained as we do not rely on "pipelines" (a pipeline in NetBind is a set of assembly-based components that are initially connected and executed subsequently). Our component model loads, connects and executes units of single components instead of a set of them. As illustrated in figure 6, each component in Microcode can be composed of four threads that run in parallel.

The *controller* component contains information about the component configuration. It has the reflective capabilities to inspect and re-configure the constituents of the CF. The inspection and re-configuration mechanisms are achieved by the OpenCOM meta-interfaces implemented on each CF (see figure 2). Information on each CF is provided to the *controller* implemented on a CF hierarchically superior to the inspected CF. As pointed out before, a CF may be composed of other CFs in order to build a composite of CFs.

5.3 An Application Scenario

In order to validate our component model, we are now experimenting with a configuration of the Router CF which uses our globally applied component model across several layers of the IXP1200 programmable router environment. This scenario demonstrates how (re-)configuration and reflection can be used to extend the network service on a router at run-time. In particular, the example emphasizes the advantage of having a single component model, which allows configuration and (re-)configuration of the service components across several layers of the programmable network design space and across different processing hardware (i.e., network processors, PCs, etc.). It also shows how (re-)configuration can be carried out in dimensions that do not have to have been foreseen when the system was designed.

The Router CF configuration illustrated in Figure 7 is a typical configuration for an IP router. It consists of several low-level, in-band components on the "fast-path" of the router, namely a classifier and

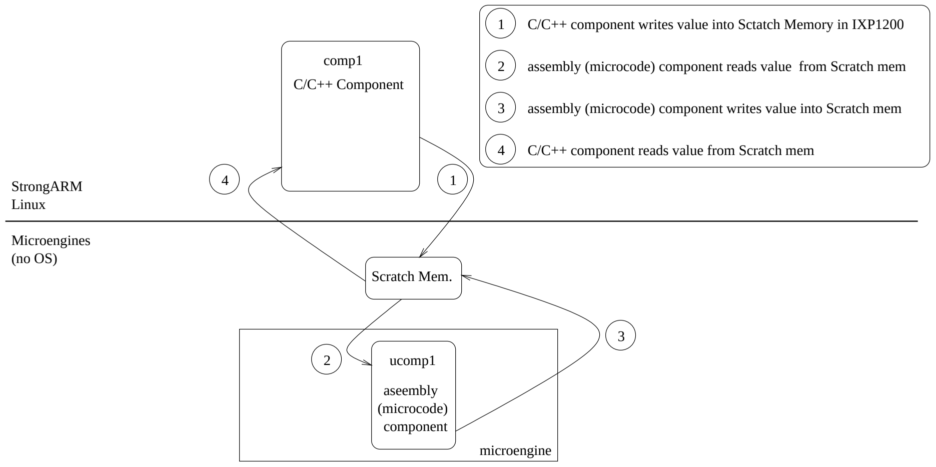
1 C/C++ component writes value into Sctatch Memory in IXP1200
2 assembly (microcode) component reads value from Scratch mem
3 assembly (microcode) component writes value into Scratch mem
4 C/C++ component reads value from Scratch mem
comp1
C/C++ Component
StrongARM
Linux
Microengines
(no OS)
Scratch Mem.
ucomp1
aseembly
(microcode)
component
microengine

Figure 5: Binding between C/C++ components and assembly (microcode)-based components

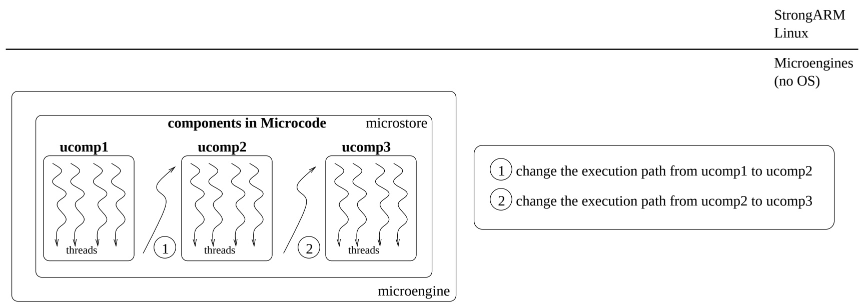

Figure 6: Binding between assembly (microcode)-based components

a forwarder, as well as a queuing and scheduling component, an application service-level component for the processing of IP options on the "slow-path", and a high-level Router CF in the control plane of the router (coordination stratum).

In the case of our IPX1200 based router architecture, we develop the above mentioned functionalities on different strata (in-band, application service and coordination layer) of our multi-strata programmable router architecture, but as component of the same component framework. In order to leverage the processing capabilities and performance of the different hardware layers, we target the different functionalities on the processing hardware best suited for them: thus, we implement the "fast-path" components in machine code for the microengines of the IPX1200 network processors. And for the IP options component on the routers "slow-path", we target on the Linux OS running on the StrongARM processor. And finally, the Routing component, which encompasses several different routing protocol components, we implement on the PC platform.

In order to illustrate the extensibility and flexibility of our approach, we include an IPv6 to IPv4 protocol translation component, which is added to the initial Router CF application at run-time. Such dynamic extensibility can be required to adapt a network environment to provide IPv6 support without restarting the network device.

Like the Router CF itself, the IPv6 to IPv4 translator is spread across different strata and thus areas of the IPX1200 router architecture. While the actual protocol translation takes place in the application services layer, the management component is established at the control plane (coordination stratum) of the router platform. The Translator is integrated with the existing functionality on the router through composition of an overall service CF. The controller component of the overall service CF integrates the

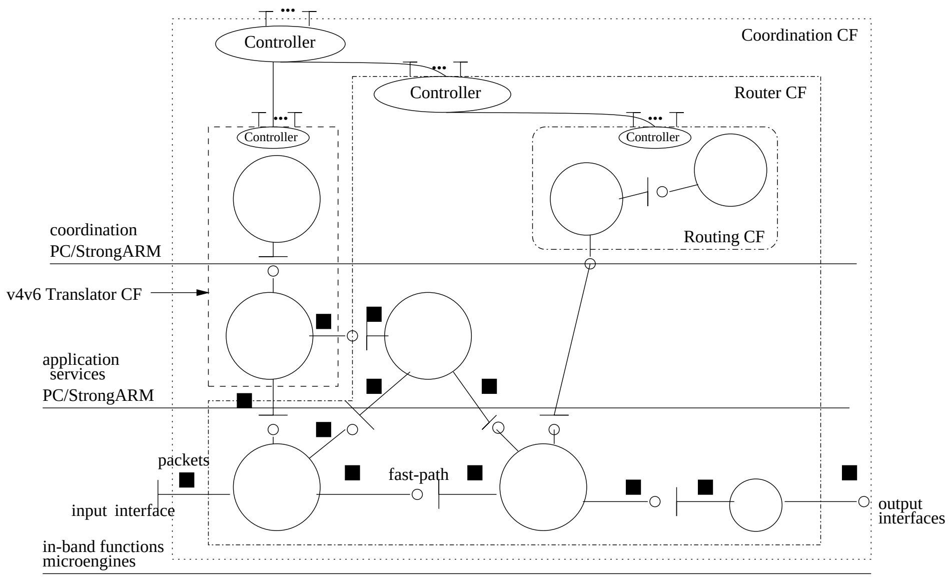

Figure 7: IPv4v6 translator application scenario

Translator by loading the composite and establishing the bindings with the Router CF interfaces.

As mentioned before, reflective techniques are used to accomplish this functionality (i.e. configuration and (re-)configuration of component-based services). Also, it is important to emphasize again that it is the uniform component model that enables components of different strata to interface with each other.

In terms of performance, we point out that the overhead seen by packets transversing the router is no more than would be incurred by Intel's standard software development environment (e.g. using Microace - Intel's component model for IXP1200 platform [13]) as the same bindings (i.e. linkages between software modules) are used in both case. Moreover, the Intel environment (Microace) is static [5], whereas the OpenCOM approach is soft and flexible.

6 Conclusions

In this paper we have proposed a component model that can potentially be applied at all all strata of the design space of programmable network platforms to create services by loading and binding involved components. We believe that the combination of components along with reflection and CFs offers a promising mechanism to configure and re-configure services in networking environments. A key strength of this model is the uniform framework to load and bind both high-level and low-level components. Therefore, loaded and bound components may reside in all strata of the programmable networking environment. As a consequence, we argue that our model facilitates fundamental re-configuration on programmable routers and hence greatly increases flexibility.

Furthermore, it is important to emphasize that our framework facilitates the extensibility and programmability of network processor based systems. These architectures are usually extremely complex and difficult to program and, as a consequence, re-configuration is hardly considered on these "primitive" environments. However, the provision of a generic framework for these architectures gives the programmer a friendly interface (abstraction) to create and consequently re-configure services based on low-level components. We do this by creating an "illusion" for the component developer that low-level components can be loaded and bound in the same way as high-level components.

7 Acknowledgements

Jó Ueyama would like to thank the National Council for Scientific and Technological Development (CNPq - Brazil) for sponsoring his scholarship at Lancaster University (Ref. 200214/01-2). The authors would like to thank the anonymous reviewers for helping to improve this paper. Finally, we would like to thank Paul Grace for his comments on this paper.

References

[1] ANTS. The ants toolkit. `http://www.cs.utah.edu/flux/janos/ants.html`, 2001.

[2] R. Braden, T. Faber, and M. Handley. From Protocol Stack to Protocol Heap — Role-Based Architecture. In *ACM SIGCOMM Computer Communication Review*, volume 33, No 1, January 2003.

[3] K. Brown. Building a Lightweight COM Interception Framework Part 1: The Universal Delegator. *Microsoft Systems Journal*, January 1999.

[4] A. Campbell, Meer G., M. Kounavis, K. Miki, J. Vicente, and D. Villela. The Genesis Kernel: A virtual network operating system for spawning network architectures. In *OPENARCH'99 - Open Architecture and Networking Programming*, New York, USA, March 1999.

[5] A.T. Campbell, M.E. Kounavis, D.A. Villela, J.B. Vicente, H.G. de Meer, K. Miki, and K.S. Kalaichelvan. NetBind: A Binding Tool for Constructing Data Paths in Network Processor-based Routers. In *5th IEEE International Conference on Open Architectures and Network Programming (OPENARCH'02)*, June 2002.

[6] P. Chandra, A. Fisher, C. Kosak, T.S.E. Ng, P. Steenkiste, E. Takahashi, and H. Zhang. Darwin: Customizable Resource Management for Value-added Network Services. In *6th IEEE Intl. Conf. on Network Protocols (ICNP 98)*, Austin, Texas, USA, October 1998.

[7] M. Clarke, G.S. Blair, G. Coulson, and N. Parlavantzas. An Efficient Component Model for the Construction of Adaptive Middleware. In *Proceedings of the IFIP/ACM Middleware 2001*, Heidelberg, November 2001.

[8] D. Comer. *Network Systems Design using Network Processors.* Prentice Hall, 2003.

[9] G. Coulson, G. Blair, T. Gomes, A. Joolia, K. Lee, J. Ueyama, and Y. Ye. Position paper: A Reflective Middleware-based Approach to Programmable Networing. In *ACM/IFIP/USENIX International Middleware Conference*, Rio de Janeiro, Brazil, June 2003.

[10] G. Coulson, Blair G.S., M. Clarke, and N. Parlavantzas. The Design of a Highly Configurable and Reconfigurable Middleware Platform. *ACM Distributed Computing Journal*, 15(2):109–126, April 2002.

[11] J. Dowling and V. Cahill. The K-Component Architecture Meta-Model for Self-Adaptive Software. In *Reflection 2001*, Kyoto, Japan, September 2001. LNCS 2192.

[12] J.P. Fassino, J.B. Stefani, J. Lawall, and G. Muller. THINK: A Software Framework for Component-based Operating System Kernels. In *USENIX 2002 Annual Conference*, June 2002.

[13] Intel. Intel IXP1200. `http://www.intel.com/IXA`, 2002.

[14] R. Isaacs and I. Leslie. Support for Resource-Assured and Dynamic Virtual Private Networks. In *JSAC Special Issue on Active and Programmable Networks*, 2001.

[15] S. Karlin and L. Peterson. VERA: An Extensible Router Architecture. In *4th International Conference on Open Architectures and Network Programming (OPENARCH)*, April 2001.

[16] R. Morris, Kohler E., J. Jannoti, and M. Kaashoek. The Click Modular Router. In *17th ACM Symposium on Operating Systems Principles (SOSP'99)*, Charleston, SC, USA, December 1999.

[17] A. Reid, M. Flatt, L Stoller, J. Lepreau, and E. Eide. Knit: Component Composition for Systems Software. In *Proc. of the 4th Operating Systems Design and Implementation (OSDI)*, pages 347–360, October 2000.

[18] S. Schmid. *A Component-based Active Router Architecture.* PhD thesis, Lancaster University, `http://www.mobileipv6.net/~sschmid/PhD_Thesis.ps`, December 2002.

[19] S. Schmid, T. Chart, M. Sifalakis, and A Scott. Flexible, Dynamic, and Scalable Service Composition for Active Routers. In *IWAN 2002 IFIP-TC6 4th International Working Conference*, volume 2546, pages 253–266, Zurich, Switzerland, December 2002.

[20] N. Shah, W. Plishker, and K. Keutzer. NP-Click: A Programming Model for the Intel IXP1200. In *2nd Workshop on Network Processors (NP-2) at the 9th International Symposium on High Performance Computer Architecture (HPCA-9), Anaheim, CA*, February 2003.

[21] C. Szyperski. *Component Software: Beyond Object-Oriented Programming.* Addison-Wesley, second edition, 2002.

[22] A. Villazón. A Reflective Active Network Node. In *IWAN*, pages 87–101, 2000.

Why is Service-Orientation Necessary for Event Correlation?

Andreas Hanemann and David Schmitz
Munich Network Management Team

Leibniz Supercomputing Center, Barer Str. 21
80333 Munich, Germany
{hanemann, schmitz}@lrz.de

Abstract. The paradigm shift from *device-oriented* to *service-oriented* management has also implications to the area of event correlation. Today's event correlation addresses mainly the correlation of events as reported from management tools. However, the correlation of trouble reports from users needs to be addressed as well, because different reports could have the same cause. In such a case the reports could be linked together and a processing has to performed only once. Therefore the response time for trouble reports could be improved and service level guarantees could be kept with less effort. We refer to such a type of correlation as service-oriented correlation. In this paper we give an overview about today's event correlation and present a selection of the applied techniques and their characteristics. Starting from a scenario for current and future distributed service provisioning we show which issues could be covered by the service-oriented event correlation. In the last section we present how we are going to approach these issues in our PhD theses.

1 Introduction

In huge networks a single fault can cause a burst of failure events. To handle the flood of events and to find the root cause of a fault, event correlation approaches like rule-based reasoning, case-based reasoning or the codebook approach have been developed. The main idea of correlation is to condense and structure events. Until now, these approaches address primarily the correlation of events as reported from management tools or devices.

In this paper we define a *service* as a set of *functions* which are offered by a *provider* to a *customer* at a *customer provider interface.* A *service level agreement (SLA)* is a contract between customer and provider about guaranteed service performance.

As in today's IT environments the offering of such services with an agreed service quality becomes more and more important, this change also affects the event correlation. To avoid SLA violations it is especially important for service providers to identify the root cause of a fault in a very short time or even act proactively. The latter refers to the case of recognizing the influence of a device breakdown on the offered services. As in this scenario the knowledge about services and their SLAs is used we call it *service-oriented.* It can be addressed from two directions.

- Top-down perspective: Several customers report a problem in a certain time interval. Are these trouble reports correlated?
- Bottom-up perspective: A device (e.g., router, server) breaks down. What services, and especially what customers, are affected by this fault?

The rest of the paper is organized as follows. In section 2 we describe how event correlation is performed today and present a selection of the state-of-the-art event correlation techniques. Section 3 describes the motivation for service-oriented event correlation and its benefits. The last section concludes the paper and presents future work.

2 Today's Event Correlation Techniques

In [1] the task of event correlation is defined as "a conceptual interpretation procedure in the sense that a new meaning is assigned to a set of events that happen in a certain time interval". We can distinguish between three aspects for event correlation.

functional aspect: The correlation focuses on functions which are provided by each network element. It is also regarded which other functions are used to provide a specific function.

topology aspect: The correlation takes into account how the network elements are connected to each other and how they interact.

time aspect: When explicitly regarding time constraints, a start and end time has to be defined for each event. The correlation can use time relationships between the events to perform the correlation. This aspect is only mentioned in some papers [1], but it has to be treated in an event correlation system.

In the event correlation it is also important to distinguish between the knowledge acquisition/representation, and the correlation algorithm. Examples of approaches to knowledge acquisition/representation are Ensel's dependency detection by neural networks [2] and Gruschke's dependency graphs [3]. There is also an approach to manage service dependencies with XML and to define a resource description framework [4].

In the past event correlation systems were focused on errors in *network elements* which are real or virtual hardware or software entities. We call this kind of correlation *device-oriented.* In contrast, *service-oriented* event correlation also regards services which are provided using network elements.

To get an overview about device-oriented event correlation a selection of several event correlation techniques being used for this kind of correlation is presented.

Model-based reasoning: *Model-based reasoning (MBR*[1]*)* [6, 5, 7] represents a system by modeling each of its components. A model can either represent a

[1] An example system for MBR is NetExpert[10] from OSI which is a hybrid MBR/RBR system. In 2000 OSI was acquired by Agilent Technologies.

physical entity or a logical entity (e.g. LAN, WAN, domain, service, business process). The models for physical entities are called *functional model*, while the models for all logical entities are called *logical model.* A description of each model contains three categories of information: attributes, relations to other models, and behavior. The event correlation is a result of the collaboration among models.

As all components of a network are represented with their behavior in the model, it is possible to perform simulations to predict how the whole network will behave.

A comparison in [7] showed that a large MBR system is not in all cases easy to maintain. It can be difficult to appropriately model the behavior for all components and their interactions correctly and completely.

Rule-based reasoning: *Rule-based reasoning (RBR[2])* [5, 6] uses a set of rules for event correlation. The rules have the form *conclusion* **if** *condition.* The condition uses received events and information about the system, while the conclusion contains actions which can either lead to system changes or use system parameters to choose the next rule.

An advantage of the approach is that the rules are more or less human-readable and therefore their effect is intuitive.

In the literature [7, 8] it is claimed that RBR systems are classified as relatively inflexible. Frequent changes in the modeled IT environment would lead to many rule updates. These changes would have to be performed by experts as no automation has currently been established. Information about the network topology which is needed for the event correlation is not used explicitly, but is encoded into the rules. This intransparent usage would make rule updates for topology changes quite difficult. The *system brittleness* would also be a problem for RBR systems. It means that the system fails if an unknown case occurs, because the case cannot be mapped onto similar cases. The output of RBR systems would also be difficult to predict, because of unforeseen rule interactions in a large rule set. According to [5] a RBR system is only appropriate if the domain for which it is used is small, nonchanging, and well understood.

Codebook approach: The *codebook approach* [3] [11, 12] has similarities to RBR, but goes one step further and encodes the rules into a correlation matrix.

The approach starts using a dependency graph with two kinds of nodes for the modeling. The first kind of nodes are the faults (denoted as problems in the cited papers) which have to be detected, while the second kind of nodes are observable events (symptoms in the papers) which are caused by the faults or other events. The dependencies between the nodes are denoted as directed edges. It is possible to choose weights for the edges, e.g. a weight for the probability that fault/event A causes event B. Another possible weighting could indicate time dependencies. There are several possibilities to reduce

[2] The GTE IMPACT system [1] is an example of a rule-based system. It also uses MBR. GTE has merged with Bell Atlantic in 1998 and is now called Verizon [9].

[3] SMARTS InCharge [11, 13] is an example of such a correlation system.

the initial graph. If e.g. a cyclic dependency of events exists and there are no probabilities for the cycles' edges, all events can be treated as one event. After a final input graph is chosen, the graph is transformed into a correlation matrix where the columns contain the faults and the rows contain the events. If there is a dependency in the graph, the weight of the corresponding edge is put into the according matrix cell. In case no weights are used the matrix cells get the values 1 for dependency and 0 otherwise. Afterwards, a simplification can be done, where events which do not help to discriminate faults are deleted. There is a trade-off between the minimization of the matrix and the robustness of the results. If the matrix is minimized as much as possible, some faults can only be distinguished by a single event. If this event cannot be reliably detected, the event correlation system cannot discriminate between the two faults. A measure how many event observation errors can be compensated by the system is the Hamming distance. The number of rows (events) that can be deleted from the matrix can differ very much depending on the relationships [5].

The codebook approach has the advantage that it uses long-term experience with graphs and coding. This experience is used to minimize the dependency graph and to select an optimal group of events with respect to processing time and robustness against noise.

A disadvantage of the approach could be that similar to RBR frequent changes in the environment make it necessary to frequently edit the input graph.

Case-based reasoning: In contrast to other approaches *case-based reasoning (CBR*[4]*)* [14, 5] systems do not use any knowledge about the system structure. The knowledge base saves cases with their values for system parameters and successful recovery actions for these cases. The recovery actions are not performed by the CBR system in the first place, but in most cases by a human operator.

If a new case appears, the CBR system compares the current system parameters with the system parameters in prior cases and tries to find a similar case. To find such a match it has to be defined for which parameters the cases can differ or have to be the same. If a match is found, a learned action can be performed automatically or the operator can be informed with a recovery proposal.

An important advantage of this approach is that the ability to learn is included in the approach. This feature is important for rapid changing environments.

There are also difficulties when applying the approach [5]. The fields which are used to find a similar case and their importance have to be defined appropriately. If there is a match with a similar case, an adaptation of the previous solution to the current has to be found.

[4] An example system for CBR is SpectroRx from Cabletron Systems. The part of Cabletron that developed SpectroRx became an independant software company in 2002 and is now called Aprisma Management Technologies [15].

In this section four event correlation approaches were presented which have evolved into commercial event correlation systems. The correlation approaches have different focuses. MBR mainly deals with the knowledge acquisition and representation, while RBR and the codebook approach propose a correlation algorithm. The focus of CBR is its ability to learn from prior cases.

3 Motivation of Service-Oriented Event Correlation

Fig. 1 shows a general service scenario upon which we will discuss the importance of a service-oriented correlation. Several services like SSH, a web service or a video conference service are offered by a provider to its customers at the customer provider interface. A customer can allow several users to use a subscribed service. The quality and cost issues of the subscribed services between a customer and a provider are agreed in SLAs. On the provider side the services use subservices for their provisioning. In case of the services mentioned above such subservices are DNS, proxy service and IP service. Both services and subservices depend on resources upon which they are provisioned. As displayed in the figure a service can depend on more than one resource and a resource can be used by one or more services.

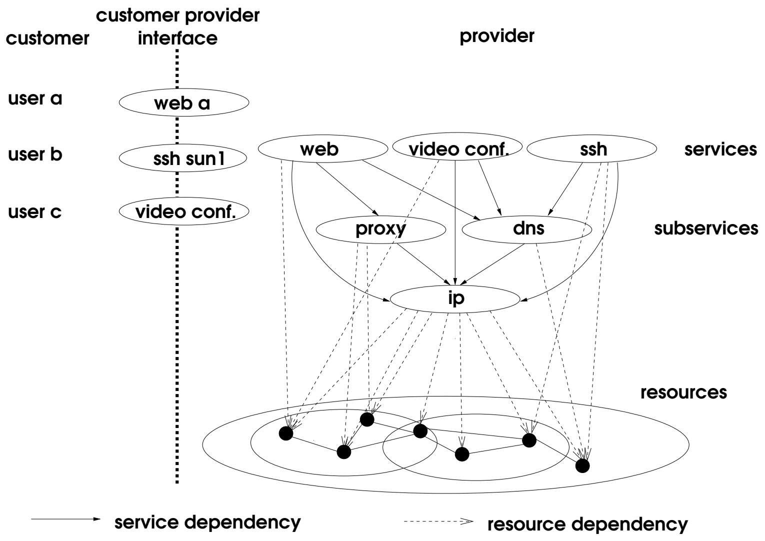

Fig. 1. Scenario

To get a common understanding, some important terms are defined in the following. We distinguish between different types of events:

Resource event: We use the term *resource event* for network events and system events. A network event refers to events like `node up/down` or `link` `up/down` whereas system events refer to events like `server down` or `authentification failure`.

Service event: A *service event* indicates that a service does not work properly. A trouble ticket which is generated from a customer report is a kind of such an event. Other service events can be generated by the provider of a service, if the provider himself detects a service malfunction.

In such a scenario the provider may receive trouble tickets from customers which indicate that the SSH, web service and video conference service are not available. When referring to the service hierarchy, the provider can conclude in such a case that all services depend on DNS. Therefore it seems more likely that a common resource which is necessary for this service does not work properly or is not available than to assume three independent service failures. In contrast to a resource-oriented perspective where all of the trouble tickets would have to be processed separately, the trouble tickets can be linked together. Their information can be aggregated and processed only once. If e.g. the problem is solved, one common message to the customers that their services are available again is generated and distributed by using the list of linked trouble tickets. This is certainly a simplified example. However, it shows the general principle of identifying the common subservices and common resources of different services.

It is important to note that the service-oriented perspective is needed to integrate service aspects, especially QoS aspects. One example of such an aspect could be that a fault (e.g., breakdown of a device) does not lead to a total failure of a service, but its QoS parameters, respectively agreed service levels, at the customer-provider interface are not met. This is also the case if a degradation in service quality is caused by high traffic load on the backbone. In the resource-oriented perspective it would be possible to define events which indicate that there is a link usage higher than a threshold, but no mechanism has currently been established to find out which services are affected and whether a QoS violation occurs.

To summarize, the reasons for the necessity of a service-oriented event correlation are the following:

– The time interval between the first symptom (recognized either by a provider, network management tools, or customers) that a service does not perform properly and the problem solution needs to be minimized. This is especially needed with respect to SLAs.
– If several user trouble reports are symptoms of the same fault, fault diagnosis should be performed only once. If the fault has been repaired, the affected customers should be informed about this automatically.
– In case of a fault in the resource layer, its influence on the associated services and customers in terms of an impact analysis needs to be recognized as soon as possible in order to take appropriate actions. By recursively following the dependencies from a resource to services an analysis which services would be affected if the resource fails could be performed. This analysis is useful to identify critical resources whose failure would have a large impact.

A precondition for a service-oriented correlation is an adequate model of services, subservices and their provision on the resources.

4 Conclusion and Future Work

In this paper we presented the difference between device-oriented event correlation and service-oriented event correlation. For the device-oriented event correlation a selection of the common techniques which are used today was presented. Their advantages and disadvantages were discussed. In the presentation of the service-oriented event correlation it was demonstrated which kind of challenges could be solved if a service-oriented event correlation would be added.

During our PhD theses we are going to approach the service-orientation from two directions (see Figure 1).

top-down perspective: If a failure of a service is detected by a customer, we want to find out which resource has caused the failure. To answer this question we start at the customer's service and track the error down the service hierarchy until a resource that causes the failure is found. During the search other service events and also resource events are used.

bottom-up perspective: Here our aim is to find out what impact a failure of a resource could have onto the provided services. Therefore we take the other way and track the service hierarchy from bottom to top. This analysis can help to identify critical resources and gives hints for future resource planning.

During our work we plan to use the MNM service model [16] which is a generic service management model and its extension by Dreo [17] to derive a service model appropriate for the service-oriented event correlation. We are also going to analyze which resource representation is well-suited for this purpose. One possibility could be the Common Information Model.

When having a look onto the correlation techniques which were presented in the paper our approach has similarities to MBR as we try to model the services and their behavior. For the correlation algorithm either RBR or the codebook approach could be used. It can be assumed that these approaches will only show a good performance if the modeling can be adequately solved. To reduce the effort for rule updates which could reduce the scalability we want to try to automatically derive the rules from the service modeling and the SLAs. To deal with cases not covered by the current modeling an additional case library (CBR) could be used. Some of these cases could be used to improve the modeling.

Acknowledgments

The authors wish to thank Gabi Dreo Rodosek for helpful discussions and valuable comments on previous versions of the paper. The MNM Team, directed by Prof. Dr. Heinz-Gerd Hegering, is a group of researchers of the Munich Universities and the Leibniz Supercomputing Center of the Bavarian Academy of Sciences. Its webserver is located at `http://wwwmnmteam.informatik.uni-muenchen.de`.

References

1. Jakobson, G. and Weissman, M.: Real-time Telecommunication Network Management: Extending Event Correlation with Temporal Constraints. In Sethi, A.S., Raynaud, Y., and Faure-Vincent, F. (eds.): Proceedings of the IEEE/IFIP Fourth International Symposium on Integrated Network Management, pages 290-301, Chapman and Hall, May 1995.
2. Ensel, C.: New Approach for Automated Generation of Service Dependency Models. In Network Management as a Strategy for Evolution and Development; Second Latin American Network Operation and Management Symposium (LANOMS 2001), IEEE Publishing, IEEE, Belo Horizonte, Brazil, August, 2001.
3. Gruschke, B.: Integrated Event Management: Event Correlation using Dependency Graphs. In Proceedings of the 9th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM 98), Newark, DE, USA, October, 1998.
4. Ensel, C., Keller, A.: An Approach for Managing Service Dependencies with XML and the Resource Description Framework. Journal of Network and Systems Management, 10(2), June, 2002.
5. Lewis, L.: Service Level Management for Enterprise Networks. Artech House, Inc. 1999. ISBN 1-58053-016-8.
6. Jakobson, G., and Weissman, M.D.: Alarm Correlation. IEEE Network, pages 52-59, Nov. 1993.
7. Wietgrefe, H., Tuchs, K.-D., Jobmann, K., Carls, G., Froelich, P., Nejdl, W., and Steinfeld, S.: Using Neural Networks for Alarm Correlation in Cellular Phone Networks. International Workshop on Applications of Neural Networks to Telecommunications (IWANNT), May 1997.
8. Appleby, K., Goldszmidt, G., and Steinder, M.: Yemanja - A Layered Event Correlation Engine for Multi-domain Server Farms. In: Pavlou, G., Anerousis, N., and Liotta, A. (eds.): Integrated Network Management, VII, pages 329-344, IEEE/IFIP, May 2001.
9. http://www.verizon.com
10. http://www.agilent.com/comms/OSS
11. Kliger, S., Yemini, S., Yemini, Y., Ohsie, D., and Stolfo, S.: A Coding Approach to Event Correlation. In: Integrated Network Management IV, pages 266-277, Chapman & Hall, 1995.
12. Yemini, S.A., Kliger, S., Mozes, E., Yemini, Y., and Ohsie, D.: High Speed and Robust Event Correlation. IEEE Communications Magazine, pages 82-90, Volume 34, Issue 5, May 1996.
13. http://www.smarts.com
14. Lewis, L.: A Case-based Reasoning Approach for the Resolution of Faults in Communication Networks. In: H.-G. Hegering and Y. Yemini (eds.): Integrated Network Management, III (C-12), Elsevier Science Publishers B.V. (North-Holland), 1993.
15. http://www.aprisma.com
16. Garschhammer, M., Hauck, R., Hegering, H.-G., Kempter, B., Langer, M., Nerb, M., Radisic, I., Rölle, H., and Schmidt, H.: Towards generic Service Management Concepts - A Service Model Based Approach. In: Pavlou, G., Anerousis, N., and Liotta, A. (eds.): Integrated Network Management, VII, pages 719-732, IEEE/IFIP, May 2001.
17. Dreo Rodosek, G.: A Generic Model for IT Services and Service Management. In: Goldszmidt, G. and Schönwälder, J. (eds.): Integrated Network Management VIII, pages 171-184, Kluver Academic Publishers, March 2003.

Aspect Testing Framework

Daniel Hughes, Philip Greenwood, and Lynne Blair

Computing Department, Lancaster University,
Lancaster, LA1 4YR. UK
d.r.hughes@lancaster.ac.uk | p.greenwood@lancaster.ac.uk | lb@comp.lancs.ac.uk

Abstract. Testing is a vital stage in the development cycle of any application but is often neglected due to the difficulty and expense to perform the stage successfully. This is especially so for distributed applications due to the co-ordination required to successfully test several distributed components simultaneously. This position paper proposes a framework implemented using Aspect-Oriented Programming and Reflection, which aims to ease the testing of such systems and other varieties of systems which encounter similar problems. Two case-studies are examined to illustrate how the framework will simplify the testing stage.

1. Background

Software testing is an important part of the development process; however, it is both difficult and time consuming and therefore often goes neglected. Distributed software engineering is one of the most high pressure areas of software engineering. This is due to the increased complexity of developing systems that run in distributed (and often unpredictable) environments, coupled with software development cycles which are increasingly restricted by the need to get products to market quickly. This often leads to sloppy testing procedures and low quality products. Distributed software engineering poses some specific problems:

- Monitoring of many distributed components running simultaneously.
- Insertion and removal of custom monitoring code.
- Reuse of testing code in other applications

An Aspect-Oriented Testing Framework will be proposed in this document, which aims to address these problems. This system should support the creation and maintenance of a distributed testing environment; allowing the behaviour of processes, which may potentially be running on remote machines, to be monitored from a central interface.

In order to ensure high quality distributed software, testing in a genuinely distributed environment is often necessary. This is because it is usually not possible to simulate enough nodes on a single machine due to the high CPU usage and network intensive nature of many distributed applications. This problem will only increase as the number of users participating in distributed communities grows. To thoroughly test the effectiveness of distributed systems for such communities, testing with ever greater numbers of nodes will be required.

Manual creation and maintenance of such tests is an extremely time consuming activity, especially where nodes are required to change their behaviour dynamically. This often requires the hand-coding of system monitoring programs to record the behaviour of nodes from a central point. Furthermore, communications code is often spread throughout a distributed application making the addition of monitoring code extremely time consuming and error-prone. To facilitate the efficient insertion and removal of monitoring to accomplish this, we expect to use a combination of Reflection [7] and Aspect-Oriented Programming (AOP) [2].

2. Aspect-Oriented Programming and Reflection

AOP is an emerging programming paradigm which extends Object-Oriented Programming (OOP) and claims to improve certain areas where OOP fails. The purpose of OOP is to allow the programmer to cleanly capture a single piece of functionality or concern in an encapsulated object, only exposing features via an interface. Suppose however that a concern can not be cleanly captured in a single object, this would normally result in the concern being spread out over several objects with sections of code implementing the concern contained in each of these objects. This leads to several problems: the code is less maintainable, the readability of the code is diminished and the encapsulation provided by OOP is lost.

AOP aims to solve these problems by allowing these *crosscutting* concerns to be cleanly captured in one self-contained unit of code. Concerns are implemented in AOP by using units of code called *aspects*. Aspects contain pieces of code called *advice* which are used to implement the crosscutting concern and the places where the advice should be applied to the OOP base-code are defined using *joinpoints*. A *weaver* is used to combine the AOP code with base-code so the appropriate links can be inserted at the places within the base-code specified by the joinpoints to reference the appropriate aspect-code. Typical examples of such crosscutting concerns which could be implemented using AOP are: security, synchronisation, and tracing. This paper will concentrate on implementing a tracing concern used for system monitoring.

When tracing a piece of code, a programmer normally inserts a number of print statements to trace the flow of the program and they can output them to screen, save them to a file or send them to a central server (if it is a distributed application). However, this can be both time consuming and unreliable since some statements could be missed from a vital section of code. Furthermore, once the tracing has been completed the print statements need to be removed which again is time consuming and potentially dangerous as deconstructive changes are being made to the code. Implementing a tracing concern such as this, using AOP, will allow the trace statements to be easily added and then later removed due to the ability to easily weave and un-weave the aspect code without needing to modify the base-code.

AOP Languages

There are several AOP languages available which are compatible with Java (our preferred development language) such as AspectJ [5] [6], JAC [10] and Hyper/J [9]. AspectJ is a relatively simple language extension to Java which uses static weaving; when the aspects are weaved with the base-code at compile time. JAC is similar to AspectJ in that it uses similar concepts but does not implement a language extension and uses dynamic weaving which means the aspects can be applied while the base-code is being executed; however in this case it is an unnecessary overhead. Hyper/J is very different to the previous two languages in that it requires the structure of the program to be carefully constructed and the inheritance/interfaces of the objects to be defined thoroughly. These constraints imposed by Hyper/J make it unsuitable for our needs. AspectJ will be our chosen aspect language due to its simplistic nature, good compatibility with Java and the ease with which it allows aspects to be defined.

The joinpoint model in AspectJ allows advice to be attached to such places as method calls, method execution, method reception, field gets, field sets and exception handlers. The advice can also be specified to be executed before, after or even around these joinpoints. AspectJ introduces the concept of *pointcuts* which are collections of joinpoints. Advice defined in AspectJ is similar to Java method constructs and can be attached to pointcuts so that they are executed at the appropriate places.

One of the main problems when using AOP is identifying which joinpoints are present in a particular application. This is especially troublesome for 3rd party objects or when the source code for an application is no longer available. This problem will hamper the reuse of aspects as the joinpoints specified in the aspect may only suit one particular application and destructive changes may have to be applied to either the aspect-code or the base-code in order to allow an aspect to be compatible with other applications.

What is required is some kind of framework which is able to examine the object for which we are interested in weaving an aspect with and then to use the information gathered to customise the aspect to suit the object.

Reflection

One way which this can be achieved is by using reflection. Reflection is defined as “The capability of a system to reason about and act upon itself. A reflective system contains a representation of its own behaviour, amenable to examination and change, and which is causally connected to the behaviour it describes. Causally connected means that: changes made to the system’s self representation are immediately reflected in its actual state and behaviour and vice versa” [1]. In this instance reflection will only be used to examine and extract information regarding the structure of the objects.

Java implements its own Reflection API which makes this process much easier. The Reflection API represents the classes, objects, interfaces currently loaded in the executing JVM. Several operations are possible using the Reflection API such as: determining the class of an object, extract information about a class's methods, fields constructors, inheritance and information regarding interfaces.

As can be seen, the Reflection API is a very powerful tool and fits our need for being able to extract information about a class. What is now needed is some kind of framework which will allow this information to customise aspects in order to change the joinpoints and advice of the aspects to suit a wider range of applications.

3. Our Approach

What we propose to use, in conjunction with Reflection, is a template structure which will incorporate the AspectJ code and our own custom tags which will identify the parts of the aspects which are 'customisable'.

Tags and Templates

We anticipate that in the first instance the framework will load the object chosen by the programmer to be examined and the reflection API will be used to extract certain information about the object. This information will consist of: class types, fields, method signatures, constructors, return values and parameter lists; elements which can be used in the joinpoints of AspectJ. The list of elements found will be presented to the user where they will be able to construct the joinpoints and select which advice/template should be used to construct the joinpoints.

The benefit of using the template and tags will allow the programmer to create a normal aspect using the language extensions which AspectJ provides and then simply substitute the parts of the aspect that need to be customised to suit different applications with the appropriate tag. Additionally, the advice in the aspect can also make use of these tags to access elements of the object such as fields or parameters. For example the following creates a piece of advice which should be executed before the method call on the method foo within the class c:

```
before() : call(void c.foo()){

}
```

Suppose that the same piece of advice is going to be used in a number of applications all of which have the class c but foo has a different implementation and to reflect this is called foo2 instead. Normally the programmer would either have to maintain two copies of this aspect or continually have to make changes to switch between the versions of foo. Instead using our framework the user could simply create a template using the tag <METHODNAME> to generalise the method name and then use the framework to switch between the two.

```
before() : call(void c.<METHODNAME>){

}
```

Or to take it a stage further the classname which the method belongs to could be generalised:

```
before() : call(void <CLASSNAME>.<METHODNAME>){

}
```

The framework will use the reflection API to examine the classes and present to the user all the potential joinpoints. The programmer can then select the class, methods, fields etc. to substitute the tags with.

Work needs to be done to expand the tags used as the code samples given here are merely examples of how the tags could be used. The framework will also need to be integrated with the Java compiler in order to ease the effort of customising the aspects to suit the base-code.

The proposed system would have the following advantages over manual testing of distributed systems:

1. Automatic insertion and removal of necessary testing code.
2. Support for automated monitoring of the state of many distributed components.
3. More rapid deployment of test scenarios.
4. Easy customisation of test cases.
5. Reuse of test code.
6. Ensures the correctness of test code.

This is not the first time that Aspects have been used for a testing/tracing purpose. The Atlas project [4] found that using print-line statements or a normal debugger are not effective when debugging a servlet, but instead found that AOP was an ideal solution. In that work the tracing aspects were created manually. In contrast, the work described in this paper extends this manual approach and aims to use Aspects in the same way but to have them created automatically.

4. Examples on AGnuS and Performance

Testing "AGnuS: The Altruistic Gnutella Server"
Consider the following testing scenario for evaluating "AGnuS: The Altruistic Gnutella Server" [3] AGnuS is a specialised Gnutella node which layers content-based routing, load balancing, caching and file filtering on top of the core Gnutella Protocol. Gnutella is a decentralised file sharing protocol. Gnutella nodes perform all functions on the network; downloading and serving files and routing all messages,

unlike traditional file-sharing services such as Napster [8] which rely upon central servers to process requests. Thorough manual testing of AGnuS is difficult for the following reasons:

Difficulty of Scalability Testing
One of the key problems with decentralised peer-to-peer networks is scalability. Any small change in the Gnutella algorithm has the potential to dramatically affect the scalability of the network. For example, not implementing the time to live (TTL) value which segments the network would destroy its scalability. Without the TTL value and the resultant hop-limit, a simply 80 byte query e.g. "Grateful Dead Live" broadcast on a network of users similar in size to that which Napster supported would require more than 160MB of data to be sent over the network. In order to reveal such potentially catastrophic performance issues, it is essential to be able to simulate networks of an appropriate size. Simultaneous monitoring of the required number of nodes is not possible manually.

Understanding the Effects of Compound Node Interactions
Another significant problem in the evaluation of complex distributed systems such as AGnuS is that the emergent structure and behaviour of the network are not always predictable based on the algorithms used to generate them. User behaviour and node failure can have unpredictable effects on the network as a whole. To thoroughly test the behaviour of the network, monitoring of individual nodes or groups of nodes throughout the network may be required, this is not possible manually.

Testing the Effect of Node Failures
As nodes in a peer-to-peer network such as this one are running on normal general-purpose workstations, which are potentially insecure and fallible, node failure must be anticipated and its effects on the network analysed. As with testing compound node interactions, this is difficult as it requires the monitoring of many distributed components simultaneously.

One example where the Aspect Testing Framework could be used, to increase the efficiency of testing for AGnuS, is in testing the accuracy and performance of the Content-Based Routing System. AGnuS' Content Based Routing system parses incoming messages, categorises them according to their type and then routes them to the most appropriate area of the network: First the method **`getQueryType(String query)`** is called. This method returns a type constant. Based on the constant returned, one of the following methods is called:

```
routeAudioQuery(String query)

routeVideoQuery(String query)

routeImageQuery(String query)

routeTextQuery(String query)
```

```
  routeSoftwareQuery(String query)
```

These methods route an incoming message to the area of the network that is richest in that data type. For Example: Running getQueryType on the incoming query "Elvis MP3" would return its type as 'AUDIO'. Based on this classification, the query will be forwarded using the routeAudioQuery method, which directs it to peers known to be rich in this file type. It is not possible to manually monitor enough simultaneously executing nodes, therefore monitoring code must be inserted throughout the content based routing system for evaluation.

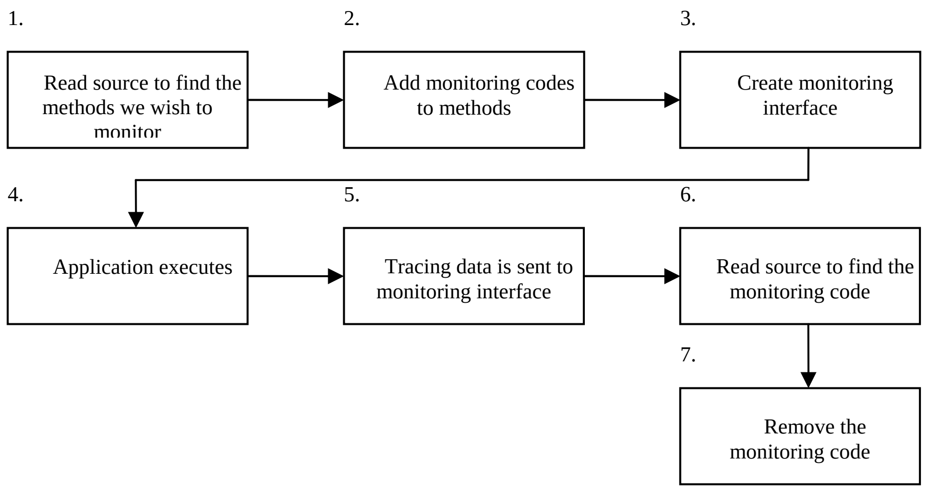
1.
Read source to find the methods we wish to monitor
2.
Add monitoring codes to methods
3.
Create monitoring interface
4.
Application executes
5.
Tracing data is sent to monitoring interface
6.
Read source to find the monitoring code
7.
Remove the monitoring code

Figure 1 Hand generation and removal of monitoring code

The proposed Aspect Testing Framework simplifies the problem of testing complex distributed systems such as AGnuS by automating the insertion/removal of monitoring code. Furthermore, the ability to easily add, modify and remove communications code makes it easy to tailor communications to fit any monitoring interface, thus facilitating the re-use of interface code.

Insertion and removal of monitoring code, especially where it needs to be spread throughout the program is a time-consuming task. The automation of this process and the re-use of monitoring code and the central monitoring interface should significantly reduce the time required to thoroughly test complex distributed systems.

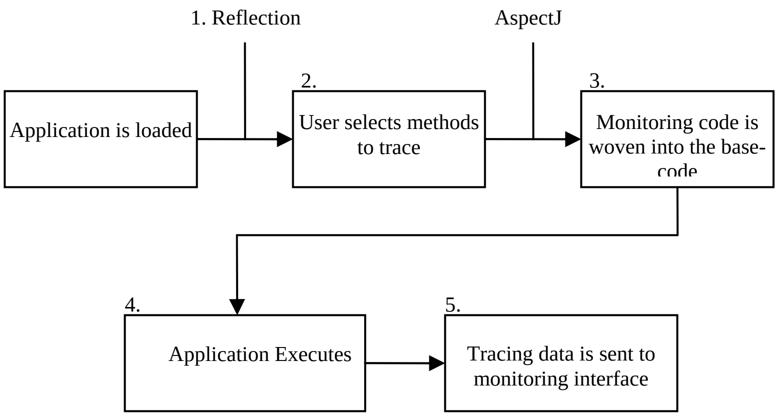
1. Reflection
AspectJ
Application is loaded
2.
User selects methods to trace
3.
Monitoring code is woven into the base-code
4.
Application Executes
5.
Tracing data is sent to monitoring interface

Figure 2 Automated insertion and removal of monitoring code

1. Java Reflection
Use of the Java reflection API allows the user to inspect the structure of AGnuS' content based routing System.

2. User Selects Methods to Trace
Presented with a list of the methods contained within the program, the user is able to select those methods which they wish to monitor.

3. Monitoring Code is Compiled into the Program
AspectJ uses the templates discussed in Section 2 to compile monitoring code into the application.

4. Program Executes
As the program executes the tracing code woven into the application sends monitoring information back to the Monitoring Interface.

5. The System Monitoring Interface
It is now possible to monitor a large number of AGnuS nodes running simultaneously, potentially on a distributed environment from a single interface.

Performance Measuring
Performance measuring is a variation of testing which is suited to be implemented using the proposed framework. Performance measuring of applications is often implemented by inserting code into an application to count the number of times a particular event occurs or to time how long a certain action takes to execute. As in the distributed application example described earlier one of the major drawbacks of performing this task is adding and removing the code to perform the measuring. Another difficulty encountered in implementing this task is customising the

performance measuring code to count and time the desired events. This is especially difficult as each different application will have different events that need measuring.

The AOP framework proposed can potentially solve both of these problems. As in the distributed application example the code required to perform the measuring can be easily added or removed due to the weaving process used by AOP. Additionally using the framework to implement this should promote the reuse of aspects which perform the measuring, as they can be easily applied to other applications by simply selecting the events via the GUI presented to the programmer which have been gathered by using Reflection. This relies on the premise that the events fit into the joinpoint model of AspectJ.

The selected elements are interpreted into joinpoints and inserted at the correct places in the aspect template. The joinpoints are attached to advice which will carry out the performance measuring such as counting or timing the selected events. The completed aspect is then woven with the base-code so that the performance measuring can take place when the application is executed and the joinpoints (which represent the selected events) are reached.

4. Summary

The Aspect Testing Framework seeks to facilitate the testing of systems by providing support for automated monitoring of the state of many (potentially distributed) components. The Aspect Testing Framework will accomplish this by automating the process of inserting and removing testing code. We present two scenarios; one derived from testing 'AGnuS: The Altruistic Gnutella Server' and a Performance Monitoring example which clearly demonstrate the potential improvements the framework provides over traditional testing methods for distributed and non-distributed software.

The Aspect Testing Framework encourages re-use of test code: the same test code and central interface can be used to monitor many different kinds of applications.

The facility to rapidly insert and remove test code throughout any application should enable more rapid creation of test scenarios; potentially alleviating the problems caused by the spiralling levels of software complexity and shrinking development schedules.

Furthermore, as test-code is compartmentalised and separate from the system which it is to monitor, it will be easier to ensure its correctness. This also aids the removal of the code from the system after testing which ensures that the correctness of the final system is not compromised by forgotten test-code.

References

[1]Coulson, G., “What is Reflection?”, http://dsonline.computer.org/middleware/, 2003.
[2] Elrad, T. et al, “Discussing aspects of AOP”, Communications of the ACM Vol. 44 No. 10 pp 33-38, 2001.
[3] Hughes, D. et al, “AGnuS: The Altruistic Gnutella Server”, proceedings of the Third international conference on peer-to-peer Computing pp202 – 203, Linköping Sweden, 2003.
[4] Kersten M., G. C. Murphy, Atlas: A Case Study in Building a Web-Based Learning Environment using Aspect-Oriented Programming. 1999.
[5] Kiczales, G. et al, “Getting Started with AspectJ”, Communications of the ACM Vol. 44 No. 10 pp 59-65, 2001.
[6] Kiczales, G. et al, “An Overview of AspectJ”, Proceedings of ECOOP pp 327-353, 2001.
[7] McCluskey G., “Using Java Reflection”, Java Developer Connection, 1998.
[8] Napster, “Napster Home Page”, http://www.napster.com, 2003.
[9] Ossher, H., Tarr, P., “Multi-Dimensional Separation of Concerns using Hyperspaces”, IBM Research Report 21452, 1999.
[10] Pawlak, R. et al, “JAC: A Flexible Solution for Aspect-Oriented Programming in Java”, Reflection 2001, 2001.

SEDAM: SErvice Discovery in MANETs exploiting Asymmetric Mobility patterns

Georg Treu

Ludwig-Maximilian University of Munich (LMU),
Department of Computer Science, Oettingenstr. 67, 80538 Muenchen, Germany
`treu@in.tum.de`
`http://wwwmnmteam.informatik.uni-muenchen.de`

Abstract. Recently, many proposals for service discovery strategies in mobile ad-hoc environments have been made. As the use of static directory services is inefficient, a number of highly dynamic mechanisms have been proposed. They address the special properties of mobile devices. However, it has not been considered so far that even in mobile ad-hoc environments, often there can be found a lot of services that run on rather immobile devices. SEDAM accounts for this mobility pattern asymmetry. Although discovered in a peer-to-peer, mobile and ad-hoc kind of way (e.g. by Bluetooth's SDP), immobile devices can also be accessed in traditional ways like for example by an IP-address. As current static directory services do not relate services with the specific situation of a Mobile Ad-Hoc Network (MANET), they are not a good alternative. Currently, a device located in a MANET is always considered as a volatile ad-hoc device. This paper shows that under certain conditions, by exploiting static properties of services found in a MANET, service discovery can be fairly optimized. Examples for such static properties can be the IP-address or the location coordinates of a device.

1 Introduction

Recently, a lot of new service discovery mechanisms have been elaborated to cope with the special constraints of mobile ad-hoc environments: memory is scarce, connections are often very unreliable and networks do not offer any specified infrastructure. Service discovery in mobile ad-hoc environments is not efficient with static directory services. Devices should be self-organised and independent, they should communicate rather on a peer-to-peer than on a client-server basis.

1.1 Related Work

There are a number of service discovery strategies that try to adopt to these constraints. SDP within Bluetooth, Sun's Java RMI based JINI, Microsoft's UPnP as well as IETF's standardized SLP. Most of these mechanisms combine peer-to-peer caching with some kind of directory agents[1] [4]. More recent

[1] Bluetooth's SDP is an exception here since it is meant to search for services only in a device's local one-hop vincinity.

service discovery strategies as e.g. GSD [1] or Allia [2] have recognized the scalability problems specialized directory agents have. By using ontology-based and bandwidth-efficient group cache and forward strategies, they offer a more sophisticated approach to service discovery in MANETs.

1.2 Mobile Clients using Static Services

Existing proposals have not considered the fact that the service offering devices found in a MANET are not always as mobile as their service searching clients. This mobility pattern asymmetry can be exploited. Many devices, such as e.g. a printer, a scanner, a fax machine or also software processes like a streaming server running on a fixed server machine can be found by mobile means, e.g. Bluetooth. Existing service discovery approaches don't consider that in many cases these static devices also provide different, more static service access points (SAPs). For instance, a printer that can be accessed by a Bluetooth interface often has an IP-address and might also be accessed via the internet. Current internet directory services can not be used here because they do not relate services with the special situation of a MANET. Another point is that it may be possible to access a printer found in a MANET via the public internet, maybe with the protection of a locally issued access token. Publicly searching internet directory services and accessing this printer as a general internet service will not be in the intention of its owner. By using Mobile-IP, even more mobile devices could be integrated with this static access method. An alternative to IP-adresses is to use location coordinates, as e.g. delivered by GPS, as a persistent access point to immobile devices. This could be done by GPSR as described in [3].
SEDAM is a service discovery strategy that applies to MANETs. It does not rely on directory agents. Devices use peer-to-peer caching to propagate service records. Peers communicate via a radio-based MANET technology, such as Bluetooth [10]. Whenever two peers come into direct radio reach they compare device profiles to determine which of their cached service records they exchange. In existing peer-to-peer caching approaches service information only propagates within one subnet. If a node leaves the subnet, service records that relate to that subnet are removed from its cache because they are not considered valid any longer. In SEDAM instead, service records can travel over network boundaries. This is achieved by distinguishing between the network technology used for service discovery from the one used for service execution. A service that is discovered by SEDAM can express its ability to be accessed via alternative URLs. A caching node keeps those alternative SAPs with a service record and can so decide in which scope the service can be used. If a service record discovered in a MANET also contains SAPs that refer to different, more static network technologies, such as an URL that can be resolved to an IP-address, that service record need not be deleted when leaving a subnet. It can be reused. With this strategy, nodes that have never been in the same subnet together can learn about each other's services. They get to know about the part of their physical environment that is not within network reach.

1.3 Example Scenario

Imagine the following real world scenario shown in Figure 1.
A digital camera is looking for a photo developing service in its physical neighbourhood. There is a photo shop offering such a service nearby, but cannot be discovered by conventional service location mechanism because the network is partitioned. At the same time, a different pedestrian is walking by the photo shop. The photo shop advertises its service to develop photos on paper via bluetooth and the pedestrian's handheld caches the shop's service offer together with the shop's official IP-address where the developing service can be accessed, too (1). As the pedestrian walks further, it gets temporarily connected with the dig-

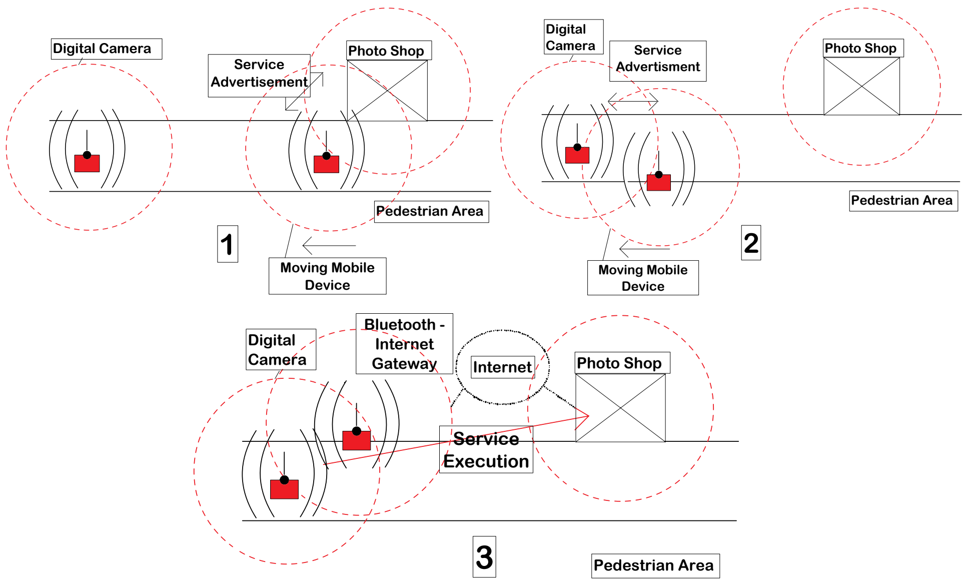

Fig. 1. A digital camera discovers a photo shop.

ital camera's bluetooth interface. It learns from the camera's profile about its need to develop photos on paper and transmits the photo shop's service offer (2). Later on, the digital camera discovers a local bluetooth-internet gateway in its bluetooth vicinity. It connects to the photo shop via internet and sends the development order. While the digital camera's owner walks down to the photo shop, the pictures are already printed out and are ready just in time (3).
With conventional service discovery strategies, the photo shop could not have been discovered easily. By the time the pedestrian's handheld got connected to the photo shop, the digital camera's network was seperated. Without the pedestrian physically walking towards the camera, it would never have learned about

the photo shop. An internet directory service, without any special positioning information, would not have been able to tell the camera about the nearby photo shop's online service.

2 Simulation

In order to analyze the proposed service discovery strategy, a java applet was developed. It implements a generic model of SEDAM and provides the user with a graphical interface to comfortably change parameters and evaluate different scenarios.

2.1 Underlying Model

The simulation's underlying model is one of a physically bounded coordinate space that contains a number of static and mobile nodes. All of these nodes can communicate directly with each other whenever they are in radio range. The exact transmission technology of the "radio" is not specified, but Bluetooth can be seen as a real world pendant to the modelled technology. Unlike the mobile nodes, all of the static nodes have an IP-address and can, additionally to their radio interface, be accessed via the Internet, which is symbolicly represented as a big square in the middle of the coordinate space (Cp. Fig. 2). Static nodes cannot move. Their position keeps fixed during the whole simulation. Mobile nodes move according to a mobility pattern that can be configured by the user. Every node, static or mobile, following a random algorithm, sets up new local services, shuts down running local services and creates new needs for services once in a while. Whenever two nodes come into direct radio range, they exchange records of the services they offer as well as records they have received from other nodes before[2]. A node's cache can hold a configurable maximum of service records. When a cache is full, old records and records that refer to mobile nodes are cleaned up first.

Service Access A service record consists of a service identifier, the coordinates of the service offering node, as well as, for static nodes, their IP-address. Dependent on a service record a node chooses from two methods to access a service. Firstly, by using a GPSR-like [3] position based routing protocol, services are accessed by the mere coordinates of their host node. This method only works if the node has not moved much since the emission of its service record and if the network is not partitioned. Secondly, if a service record contains an IP-address, the service is accessed by traditional IP-routing. Arbitrary static nodes can serve mobile nodes as an internet access point here.

[2] Profiles are not maintained by the simulation so far. Currently, two connected Nodes always exchange all of their service records.

2.2 Configuration

To dynamically evaluate different scenarios, the most important simulation parameters are configured by the user. *Simulation Speed* doesn't effect the quantitative result of a simulation run. It is just used to determine how fast the simulation runs. *Coord Width* and *Coord Height* determine the bounds of the coordinate space the nodes are located in. *Transmission Radius* relates to the radio technology used. 10 m, resp. 100 m would be typical values for Bluetooth. The overall number of nodes is set by *Node Count*, *Static Nodes* defines how many of them are static (the remaining nodes are mobile). The way the mobile nodes move is determined by *Mobility Pattern*. Currently there are two patterns implemented. *RandomMobility* corresponds to the Random Mobility Model and *RandomWaypoint* to the Random Waypoint Mobility Model. Both models are described in [9]. *Cache Size* denotes the maximum number of service records a node's cache can hold and *Service Diversity* specifies how many different services are distinguished within the system. Finally, *Discovery Time-out* specifies after how many simulation steps a node will abort the unsuccessful search for a needed service.

2.3 Analysis

The simulation's GUI contains a simulation panel on the left side of the screen, showing the movements of the nodes as well as service executions between nodes in a graphical way. See figure 2 for a screenshot. Additionally to this demonstrative view, a simulation run is analyzed more mathematically in the form of six charts on the right side of the screen. The charts display certain performance indicators measured against simulation time. They are updated constantly while the simulation takes place. *Service Executions per Step* depicts the overall number of successful service excutions per simulation step. The average time it takes a node to successfully find and execute a needed service can be seen from *Avg. Discovery Time*. If a service need can be matched to a service record but an error occurs during service execution (e.g. if the service has moved or shut down), a SAP failure is generated. *SAP Failures per Step* measures the number of those failures per simulation step. In the simulation, a service need is either executed successfully or it times out. *Success Rate* represents the fraction (in percent) of the service needs that can be successfully matched and executed. *Advertisements per Step* shows the overall number of service records that are exchanged in one simulation step. Finally, *Cache entries* counts the overall number of service records that are currently cached by the nodes.

Results The most important performance indicator for SEDAM is *Success Rate*. The higher the *Success Rate* of a service discovery, the more effective the system is. Most of the parameters presented in 2.2 have an obvious influence on the *Success Rate*. An increased *Node Count*, a bigger *Cache Size*, smaller *Coord Width* and *Coord Height*, a bigger *Transmission Radius*, a lower *ServiceDiversity*, a longer *Discovery Time-out*, all of these changes raise the rate and even without

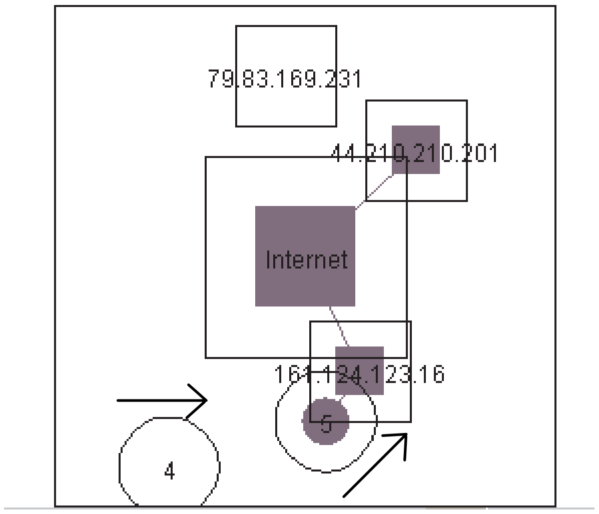

Fig. 2. A screenshot of the simulation.

any simulation it can easily be concluded why. Astonishing is that changing the mobility pattern has almost no effect on the *Success Rate*.
Although increasing *Node Count* generally raises the system's *Success Rate*, changing the fraction of *Static Nodes* for a given *Node Count* does not yield such clear results. This is due to the complex interaction between mobile and static nodes in SEDAM. An ideal system performance is achieved when there are firstly enough static nodes that offer persistent service records and secondly if there are enough mobile nodes to spread the service records. So, the most interesting point about the simulation is to determine the fraction of *Static Nodes* that yields an optimum of the *Success Rate* for a given setup. To give an example for that, another test program was inserted into the simulation. All parameters except *Static Nodes* were set to (not necessarily ideal) standard values: *Node Count* was set to 30 which appeared to be a reasonable value, *Discovery Timeout* to 50 steps, *Service Diversity* to 20, both, *Coord Width* and *Coord Height* to 100 m, *Cache Size* to 8, *Transmission Radius* to 10 m and the *Mobility Pattern* to RandomWaypoint. All values of *Static Nodes* between 5 and 25 were tested each in a 200 step simulation run. See Table 1 for a summary of the test. The maximum of the *Success Rate* for the given scenario is reached at a fraction of 20 *Static Nodes*. Changing the number of *Static Nodes* in either direction seems to lower the *Success Rate*. The simulation applet (JRE 1.2+ required) itself can be found on:
http://wwwmnmteam.informatik.uni-muenchen.de/˜buchholz/sedam/sedam.html

Table 1. Results of Simulation Test Runs with a *Node Count* of 30

Success Rate (%)	*Static Nodes*
27.35	5
26.15	10
34.17	15
44.23	20
31.81	25

3 Conclusion

It can be verified with the simulation that SEDAM is not going to work in all possible scenarios. Nor is it going to provide a reliable discovery of all the offered services in a device's environment. It is a best effort service. For a good applicability of SEDAM as a service discovery strategy there needs to be a sufficing number of nodes in a rather small area and a good mixture of static and mobile nodes. Exact values for these parameters cannot be given in general since they are dependent on various external factors. However, the simulation shows that, if conditions are good, SEDAM can achieve a highly statisfying performance (expressed as *Success Rate*).

3.1 Future Work

In the context of a master's Thesis, a prototype for SEDAM is currently being developed. The prototype will be a middleware module that can be integrated into existing applications. In the first place, it will use Bluetooth as the underlying radio technology. It will be implemented in Java and is designed to run on PCs as well as on personal digital assistants (PDAs) and other Java enabled mobile devices.

Service Matching: As can be seen from work like [1] the mechanisms that match service needs against service offers can be optimized.
Firstly, a description of a service need should be matched against a service record in a precise yet at the same time in an extendable and semantic way to maintain service offering and service needing devices independently from one another. This could be done by using an ontology language for service records. Examples for such languages can be found in [5], [6] and [7].
Secondly, service matching mechanisms can be optimized with regards to scalability. By a kind of prefiltering, exhaustive service matching can be avoided. Each device that collaborates in the service discovery should maintain a profile, as e.g. proposed in [8]. The idea is that mobile devices should only exchange those service records that match each other's profile. Like this, device profiles can serve as a filter. Unrelevant service records are no longer exchanged between

devices. Unnecessary service advertisements are avoided. Note that this filtering reduces both side effects, *Advertisements per Step*, which can be seen as a measurement for the network traffic and computing time on the nodes, as well as the number of *Cache Entries*.

***Caching Algorithms*:** Memory in mobile devices is low, so caches are small. Sophisticated caching strategies should be applied to distinguish service records that have become unrelevant from the still valid and useful ones. A caching strategy should therefore not only implement a simple LRU scheme, but evaluate service records with regard to the mobility and availibility of the service offering device. The method used to filter service records in the beginning could also be applied to calculate the relevance of cached service records to the current profile. The prototype will be structured to support different modules for ontology languages and profile creation techniques. Optimizing those modules for a more sophisticated version of SEDAM will be the subject of future work.

References

1. D. Chakraborty: GSD. A Novel Group-based Service Discovery Protocol for MANETs, MWCN, 2002 (URL: http://www.cs.umbc.edu/~dchakr1/cadip2/mypapers/MWCN/301.pdf)
2. D. Chakraborty, O. Ratsimor: Allia. Alliance-based Service Discovery for Ad-hoc Environments, ACM Mobile Commerce Workshop, 2002 (URL: http://www.csee.umbc.edu/~oratsi2/publications/AlliaMC2002/AlliaMC2002.ps)
3. B. Karp, H.T. Kung: GPSR. Greedy Perimeter Stateless Routing for Wireless Networks, Mobile Computing and Networking, 2000, pages 243-254 (URL: http://www.eecs.harvard.edu/~karp/gpsr-mobicom2000.ps)
4. C. Bettstetter, C. Renner: a Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol, EUNICE, 2000 (URL: http://www.lkn.ei.tum.de/~chris/publications/eunice2000-slp.ps.gz)
5. S. Avancha: Enhancing the Bluetooth Service Discovery Protocol, Technical report, University of Maryland Baltimore County, 2001 (URL: http://research.ebiquity.org/re/papers/enhancedsdp.pdf)
6. T. Strang, C. Linnhoff-Popien: a Context Ontology Language to Enable Contextual Interoperability, DAIS, 2003
7. D. Chakraborty, F. Perich: DReggie. Semantic Service Discovery for M-Commerce Applications, Workshop on Reliable and Secure Applications in Mobile Environment, 20th Symposiom on Reliable Distributed Systems, 2001 (URL: http://umbc.edu/papers/dreggie.pdf)
8. A. Held, S. Buchholz, A. Schill: Modeling of Context Information for Pervasive Computing Applications, SCI, 2002 (URL: http://wwwrn.inf.tu-dresden.de/~buchholz/hades/SCI2002-paper512JH.pdf)
9. V. A. Davies: Evaluating Mobility Models Within an Ad Hoc Network, Master's thesis, Colorado School of Mines, 2000 (URL: http://toilers.mines.edu/papers/pdf/vanessa-thesis.pdf)
10. Bluetooth SIG: Specification of the Bluetooth System, Volume 1, 2001 (URL: https://www.bluetooth.org/foundry/specification/document/Bluetooth_V1.1_Core_Specifications)