Project-Team PASS

Processes for Adaptive Systems

Brest

Activity Report
2013
1 Team

Team leader
Antoine Beugnard, Professor, Telecom Bretagne

Administrative assistant
Armelle Lannuzel, Telecom Bretagne

Telecom Bretagne staff
Fabien Dagnat, Associate Professor
Julien Mallet, Associate Professor
Siegfried Rouvrais, Associate Professor
Maria-Teresa Segarra, Associate Professor

PhD students
Iyass Alloush, since October 2011
Fahad Golra, SFERE grant, since December 2008
Anthony Lee, Cifre Orange Labs, since January 2011
Sébatien Martinez, Région Bretagne grant, since September 2012
HUYNH Ngoc Tho, Vietnam grant, since December 2013

Associate members
Jean-Marie Gilliot, Associate Professor, Telecom Bretagne, Associate Member
Claire Lassudrie, Associate Professor, Telecom Bretagne, Associate Member

2 Overall Objectives

2.1 Overview

Adaptive software aims at adjusting various artifacts or attributes in response to changes in the system operating environment. By operating environment we mean everything that may affect the software system behavior and its properties, for example changing requirements, execution platform, and resources. Adjustment can be performed at design time and leads to the disruption of software operation. However, the higher demand for ubiquitous, pervasive, embedded, and mobile environments, has led to the need of runtime adaptation, i.e., without software disruption. Runtime adaptation increases software complexity as the software itself has to cope with variability. Complexity is even increased if the software system is distributed. Indeed, adjusting operations may involve artifacts running on different execution platforms and adaptation mechanisms available on them may vary.

On the other hand, software complexity is traditionally tackled by focusing on the software product itself or its related design process. The first one mainly proposes artifacts such as
languages, software architectures, frameworks, and middleware that help on mastering complexity at a manageable degree for designers and maintainers. The second one is interested in providing a methodology (process) to design a software system that satisfies user requirements. It guides developers in taking into account important functional and non-functional concerns and may be reused for each development. Both areas offer different but interesting approaches that help in managing software complexity.

The PASS team aims at combining both approaches to master adaptive software complexity. More specifically, it aims at defining system and process models and their relationships in order to ease the development and the adaptability of software systems. The mottos of the PASS research are:

1. *Adaptation requires to reify the software design process and trace its enactment.*

   In order to build adaptive software, it is necessary to reify its design process, rationales behind it, and to consider adaptivity as a first class property. Simply knowing the current variant of the software and the targeted one, as in traditional reflection-based approaches or more recently in DSPLs (Dynamic Software Product Lines), is not enough. Indeed, they do not consider relevant information for adaptation such as when it is possible to change, what can be changed, and how building blocks state should be managed.

2. *Processes are software.*

   The description, specification, and design of processes should follow the same process as software \[Ost87\]. As a consequence, current standard process models such as BPEL or BPMN are inappropriate since, they poorly distinguish specification from design and enactment.

3. *Components have distributed access points.*

   Classical software component models are bound to their physical execution targets (devices) and are connected through connectors (bindings) that ensure the assembly. We propose a paradigm shift releasing the software component model dependency from its underlying infrastructure. As a consequence, the access points of a single component can be distributed among various devices and (remote) communications are managed inside components.

   Our research is intended to be *vertical* in the sense that all aspects of systems are of interest: specification, design and implementation.

   Solutions to enact these mottos are at the intersection of model driven engineering, software architecture, process modeling, verification and validation, and distributed systems. We intend to scientifically contribute to the three first domains (model driven engineering, software architecture, process modeling).

   In this scientific context, we are interested in three kinds of systems: large scale, highly dynamic (mobile, P2P) and strongly constrained (real-time, embedded) where non functional

---

properties such as safety, performance and reliability are important. For instance, ambient assisted living, satellites, cloud, mobile games, pervasive learning, etc.

On the application side, the PASS team explores the application to software systems such as satellite, home-automation, multi-users games, etc but also to non-software systems such as educational systems.

2.2 Key Issues

In its current state, PASS studies the following key issues:

- Adaptation decisions require information to trigger adaptations.

  What is the information to be traced? How the process can keep track of this information?

- Non-functional properties (e.g., security, performance, dependability, scalability) and other quality attributes can be considered as drivers for software and enterprise architecture design.

  How methods, tools, models and processes could support non-functional specifications for analysis?

- Many software component models have been developed over the years. They all rely on an explicit binding with connector.

  Can’t we provide an efficient component model that hides connectors?

- Software deployment relies on packet distribution with coarse-grain updating operators.

  Are there other levels of deployment? Are there other units of adaptation? What are the proper control primitives for updating software?

3 Scientific Foundations

3.1 Modeling and Meta-Modeling

Modeling is a central activity of science and engineering. Computer science brought the idea to think to model as object of interest. The reification of models leads to study ways models are described. In the context of software design, with the importance of the UML modeling approach, the Object Management Groups made the Model Driven Architecture (MDA) initiative. This approach is generalized to system modeling and system of system modeling. Earlier, computer scientists had defined a theory of languages and grammar, that can be compared to the model and meta-model approach of the OMG. In fact, both approaches are very close; formalisms, theory change but underlying principles are close, if not identical.

In [Mor67], W. Morris has shown that learning of models is not learning of modeling and had made suggestions to enhance the process of developing the intuitive skill associated with model building. Almost 50 years later, the understanding of models has improved thanks

to its explicitation and the development of tools. However, current approaches (UML, DSL, MOF, EMF, XML, OWL, etc.) have shown that none is sufficient and that users (scientists, engineers, developers, etc.) need flexibility, merging and preciseness. Flexibility, since existing meta-models are not offering all required concepts; as an example, UML proposes a partial solution with its profiles. Merging, since different points of view need to be combined; the proliferation of architecture framework[Web14] is an example. And Preciseness, since the interest of tools is not only drawing or writing, but its also checking, proving, testing, generating code or documentation with a controlled semantics.

3.2 Type Systems

This would be more detailed later.

3.3 Process Modeling

The description of processes is central to Software Engineering. From early life-cycle that were coarse grain descriptions to fine-grain description like BPMN, the explicitation of activities, of who is doing something and of what is produced, has always been an issue in (software) development.

The current process modeling languages appear to have two distinctive problems. First, they seem to ignore the importance of a consistent approach that handles process in all the stages of its lifecycle. Either a single process model seems to represent a process in all phases (e.g. at specification phase and the implementation phase) or it has to be transformed to a completely different approach for enactment. For example BPMN models processes in all stages of development through a single notation, however it does not offer the possibility to enact them. Consequently, process developers are bound to transform the models to BPEL for enactment. Second, most of the approaches focus on the flow of activities defining the order of (presumable) execution. Some approaches use event based mechanisms to induce reactivity, but still their focus remains on the flow of activities. Approaches like Event-driven Process Chains (EPCs) use both types of inputs for the activities (events and artifacts) together, which clutters the process model. Dataflow and control flow in a process, at the same time, makes it hard to conceptualize the interactions of an activity to its context.

Software processes just like software systems are based on the notion of lifecycle, where each stage of development has different concepts to frame. Each phase of a software process organizes different factors and issues related to the degree of its maturity in terms of completeness. Fuggetta defined a software process as, the coherent set of policies, organizational structures, technologies, procedures, and artifacts that are needed to conceive, develop, deploy, and maintain a software product. But we tend to think more like the Osterweil’s analogy of a software process, where he advocates that processes share the same nature and complexity as the software system and should be treated the same way [Ost87]. Thus we are of the view that software processes themselves need to be conceived (specification), developed (implementation), deployed (enactment) and maintained (monitoring). As the life-cycle phase of

---

a software process advances, the focus on issues related to the process also changes. For example, in specification phase the focus remains on which artifacts would be handed over by the activity. On the other hand, for enacting a process the focus shifts to when should an artifact be handed over by this activity. During the enactment phase, the reasoning about the activity is not targeted towards the choice of its inputs and outputs, rather its directed towards the related hows and whens.

**Glossary:**

**Activity** A time period dedicated to the production of Artifacts by persons playing their Roles in the Activity. An activity - as a program - has a specification, several implementations resulting from design choices, and many enactments.

**Artifact** Any human-made product resulting from an Activity. Documentation, source code, review, test plan are some examples of artifacts.

**Role** An abstraction of a position denoting a responsibility in an Activity.

**Process** An organized set of activities, roles and artifacts.

### 3.4 Software Components

Software component is an old idea. The first reference is usually attributed to M.D. McIlroy [McI69] in 1968. From that time, many component models were proposed and many surveys compare them. Components may be classified according to their lifecycle, to their interface specification, to their computational model, their implementation targets and many other dimensions.

Software components are units of composition, units of specification, units of management. As such they have their own development lifecycle/process. In [CSVC11], Crnkovic et al classify more than 20 components models with respect to their component lifecycle. It is worth noticing that these models do not propose any progression in their description. A component model is usually described at a selected level - often implementation, some time specification - with no references to its various forms from high level specification to low level implementation.

Software components as units are defined by their interface. It is now well admitted that a good way to describe these interface is through contracts [3]. However, if low level contracts (syntactic and semantics) are well established, higher levels (behavior and QoS) are seldom used.

**Keywords:** Abstraction, composability, component.

**Glossary:**

**Abstraction** a mechanism and practice to reduce and factor out details so that one can focus on few concepts at a time.

**Composition** a system design principle that deals with the inter-relationships of components. A highly composable system provides recombinant components that can be selected and assembled in various combinations to satisfy specific user requirements.

**Component** a unit of composition with contractually specified interfaces and explicit

---


context dependencies only. A software component can be deployed independently and is subject to composition by third parties.

3.5 Adaptation and reconfiguration

In this research area we are interested in providing developers with methods and tools to easily build (distributed) adaptive software. To achieve this goal, we are currently working on the definition of 1) a generic, customizable model for distributed dynamic adaptation mechanisms at the middleware level, and 2) a set of tools that may be used by adaptation designers in order to ease the customization of our model for their needs.

Models for distributed dynamic adaptation mechanisms. In a distributed component-based software, a set of components located in different nodes, cooperate in order to provide services. Making such a system adaptive requires including adaptation mechanisms that are themselves distributed. Several works exist that propose distributed algorithms for either deciding or planning changes [FDP+12]. Although those approaches contribute to simplify the task of building an adaptive software, they fail to provide an architectural view of the adaptation mechanisms. Such a view would allow designers to choose different distribution strategies according to their needs, i.e., some components may be managed in a distributed manner while others may not.

To tackle this limitation we have been working on a model for distributed and coordinated dynamic adaptation [15]. We have proposed a model that constrains designers considering what are the dynamic adaptation services they need for their application and how many instances (and location) of the selected services should be created. According to the number of instances of the different services, coordination components may be mandatory and different coordination strategies are available to be used by designers, whether for decision, planning or execution services. In order to validate this work, we considered the distributed data management application domain and we intend to dynamically change data replication policies depending on the execution environment.

Development process for adaptive distributed software. The second research area is related to the automatic generation of dynamic adaptation mechanisms: decision making, planning, and execution. This work is based on the main result of Eric Cariou and Chantal Kaboré PhD work, the Cloud Component approach [7][9], a communication abstraction (Cloud Component) used to model distributed services and a refinement process that allows for automatic generation of architectural variants. It is included in a current research trend that proposes DSPLs (Dynamic Software Product Lines) and MDE (Model-Driven Engineering) to automatically compute the actions to be performed to build the target variant when an adaptation is needed, i.e., to compute the "adaptation plan" or "reconfiguration plan". In current approaches, these actions correspond to operations on the application architecture (add

component, remove component, etc...). These works propose general enough approaches that can be applied to all model-based applications, but they fail to provide a systemic approach for automatic generation of adaptation plans.

Compared to these approaches, in this research area we are interested in investigating how to generate not only architectural modification actions but also (all the) actions of an adaptation plan including application-dependent actions. We claim that during the development process relevant information may be collected then exploited to compute adaptation plans.

We have been working on one type of application-dependent actions: those related to the data managed by the components affected by an adaptation. Indeed, when a component is modified or replaced its data should also be taken into account. These data are ignored by most of current approaches that assume stateless components. To tackle this issue, we have been working on two areas:

- The first one aims at helping a developer to explicitly add information related to the data managed by the application [10]. This information is related to the entities that manage data and the operations used to manage them. Such information is then used to compute data-related actions to be added to adaptation plans. More specifically, data transfers between components can be computed and added to the adaptation plan.

- The second one is related to the concept of variation point in DSPLs and its reification at runtime. Indeed, current approaches assume that all variation points in a DSPL of an application should be available at runtime: each time an optional feature is present in the DSPL, the application should be able to change at runtime to include or remove the feature. In this work, we claim that each variation point identified in the DSPL may be reified at runtime and we investigate a development process that allows to reify such variation points [11].

Glossary:

Adaptive software a software that has the ability to adapt at runtime to handle such things as changing user needs, system intrusions or faults, changing operational environment, and resource variability.

Dynamic or runtime adaptation a change on a software that takes place after deploying it. It includes deciding about the change to perform, planning the necessary operations on the software, and executing them.

4 Application Domains

4.1 Component Software Adaptation

Participants: Antoine Beugnard, Maria-Teresa Segarra Segarra, Julien Mallet, Fabien Dagnat, Anthony Lee, Sébastien Martinez, Tho HUYNH Ngoc.

Apply the mottos [1] [2] and [3]

Domains: Home-automation, HPC, Large scale applications.
4.2 Process Design

Participants: Antoine Beugnard, Fabien Dagnat, Fahad Golra.

Apply the mottos 1, 2.

4.3 Educational Systems

Participants: Siegfried Rouvrais, Antoine Beugnard.

An educational system provides services to students and society. For strategic alignment, to adjust in response to requirements or resources changes in the societal environment, the management of educational transformation is of decisive importance to Higher Education Institutions (HEI) and regulation bodies. During the last decade, various models of quality management have emerged in the form of quality assurance standards (e.g. accreditation bodies, European or international frameworks) that guide programme leaders, designers or deans of academics to evaluate and improve educational systems including syllabus, curricula, workforce, workspaces, support services, or administrative processes. However, to date, educational system complexity is traditionally tackled by focusing restrictively on the course or curricula contents. There is no standard commonly accepted for conceptually describing the overall complexity of educational systems, restricting the modernization agenda of HEI nationally, in Europe, and internationally.

As the functional, regulatory and quality requirements (e.g. non functional properties) of educational systems increase, it is a major challenge today to unambiguously describe and organize concepts among the various stakeholders involved in programme design or transformation, so as service consumption. The actual complexity of educational systems and the dynamic of their requirements call today for sound engineering principles. The aim is this research is to define and provide methods, processes and tools so as to design and verify educational systems that satisfy requirements under controlled resources. Based on modeling, meta-modeling and business process modeling foundations, well defined models and processes for continuous improvement could stimulate a programme reform, transformation or renewal so as to better align with requirements (e.g. learning outcomes). A system modeling approach is an obvious choice for meeting such challenges since it makes it possible to meaningfully, unambiguously, and accurately specify concepts, relations, and viewpoints among stakeholders, so as to flexibly manage change. Educational architectures and frameworks can thus be defined, with varying degrees of rigor, so as to prepare and organize various views and models for stakeholders, manage processes for the various engineering phases, and systematize some analysis for better quality fulfillment.

From this perspective, applying the mottos 1, 2 and based on a convergence of system engineering, enterprise architecture, and education engineering principles, this PASS research track focuses on architecting educational systems and expects to provide in-depth and usable models, methods, and tools to address issues of educational system design, transformation, and adaptability.
5 Software

5.1 ReCaml

Participants: Fabien Dagnat [contact point].

To fix bugs or to enhance a software system without service disruption, one has to update it dynamically during execution. Most prior dynamic software updating techniques require that the code to be changed is not running at the time of the update. However, this restriction precludes any change to the outermost loops of servers, OS scheduling loops and recursive functions. Permitting a dynamic update to more generally manipulate the program’s execution state, including the runtime stack, alleviates this restriction but increases the likelihood of type errors. ReCaml [6] is a language for writing dynamic updates to running programs that views execution state as a delimited continuation. ReCaml includes a novel feature for introspecting continuations called match cont which is sufficiently powerful to implement a variety of updating policies. We have formalized the core of ReCaml and proved it sound (using the Coq proof assistant), thus ensuring that state-manipulating updates preserve type-safe execution of the updated program. We have implemented ReCaml as an extension to the Caml bytecode interpreter and used it for several examples.

5.2 Pymoul
t

Participants: Sébastien Martinez [contact point], Fabien Dagnat.

When willing to update software at runtime, one choses a platform that will give mechanisms to modify the software during its execution. For each problematic of dynamic software updating, a given platform would offer one unique answer. This means that the choice of the update mechanisms to use for modifying a running application does not belong to the developer of the application or of its updates.

Pymoul[13]isaPythonlibrarythatenablesapplicationandupdatesdeveloper to chose the mechanisms they want to use when designing dynamically updatable applications or dynamic updates. For that purpose, Pymoul provides the mechanisms implemented in several platforms from the literature. Pymoul also proposes a generic API for designing updates independently from any platform used.

Pymoul uses a modified version of Pypy: a Python interpreter written in Python. This modified version of Pypy allows specific interpreter operations such as intercepting object creation or capturing and modifying the runtime of threads. Pymoul was tested on several test cases from other platform, proving itself to be a good platform for testing and prototyping dynamic software updates.

5.3 MetaModMap

Participants: Thang Pham Quyet [previous PhD], Antoine Beugnard [contact point].

MetaModMap is an eclipse-based plugin which allows to define the intensional semantics correspondences of elements between two similar metamodels. From this definition, the plugin
provides a facility which rewrites a legacy transformation definition for the old metamodels to a new one for the new similar metamodels.

This software is developed using Maude programming language and its extension Moment2.

5.4 Openflexo

Participants: Antoine Beugnard, Fabien Dagnat, Christophe Guychard [contact point], Sylvain Guérin [SCIC

Openflexo is an open-source (GPL3) business architecture platform that supports collaborative agile missions by seamlessly transforming multifaceted enterprise models into business oriented deliverables. Openflexo Modeller provides frequent delivery production cycles (storyboard, documentation, operational prototypes and applications). Openflexo is business oriented. It's a tool gathering all actors around a shared visualization of a multifaceted model, continuously updated and enriched at each iteration.

6 New Results

6.1 A Refinement based methodology for software process modeling

Participants: Fahad Golra

There is an increasing trend to consider the processes of an organization as one of its highly valuable asset. Processes are the reusable assets of an organization which define the procedures of routine working for accomplishing its goals. The software industry has the potential to become one of the most internationally dispersed high-tech industry. With growing importance of software and services sector, standardization of processes is also becoming crucial to maintain credibility in the market. Software development processes follow a lifecycle that is very similar to the software development lifecycle. Similarly, multiple phases of a process development lifecycle follow an iterative/incremental approach that leads to continuous process improvement. This incremental approach calls for a refinement based strategy to develop, execute and maintain software development processes.

This thesis develops a conceptual foundation for refinement based development of software processes keeping in view the precise requirements for each individual phase of process development lifecycle. It exploits model driven engineering to present a multi-metamodel framework for the development of software processes, where each metamodel corresponds to a different phase of a process. A process undergoes a series of refinements till it is enriched with execution capabilities. Keeping in view the need to comply with the adopted standards, the architecture of process modeling approach exploits the concept of abstraction. This mechanism also caters for special circumstances where a software enterprise need to follow multiple process standards for the same project.

On the basis of the insights gained from the examination of contemporary offerings in this domain, the proposed process modeling framework tends to foster an architecture that

1 Société Coopérative d’Intérêt Collectif: a social and solidarity economy company dedicated to cooperation. http://www.les-scic.coop/sites/fr/les-scic/
is developed around the concepts of “design by contract” and “design for reuse”. This allows to develop a process model that is modular in structure and guarantees the correctness of interactions between the constituent activities. Separation of concerns being the motivation, data-flow within a process is handled at a different abstraction level than the control-flow. Conformance between these levels allows to offer a bi-layered architecture that handles the flow of data through an underlying event management system. An assessment of the capabilities of the proposed approach is provided through a comprehensive patterns-based analysis, which allows a direct comparison of its functionality with other process modeling approaches.

6.2 Other results

Publications: [2, 12, 10, 17, 13, 8, 5, 6, 7, 8, 9, 11, 14, 16, 15, 4]

6.3 Assessment of achievements

Participants: All participants.

The results achieved by the PASS team must be compared with the key issues presented in the objective part. Not all key issues have deserved attention yet. However, a few of them have been sufficiently well explored to start and draw conclusions.

The first key issue is “What are the information to be trace? How the process can keep trace of these information?” The information to trace are mainly the information related to the decision choices that have an impact on the component parts that are to be adapted or modified. We experiment different models where to attach those informations. Those models are close to process models or feature models; all are some kind of process abstractions.

The second key issue is “Can we provide an efficient component model that hides connectors?” We have proposed a component model that illustrates a paradigm shift. We can develop distributed application with components that encapsulate all the communications required by the distribution, but that are composed only locally. This component model should be the beginning of new experiences and developments.

7 Contracts and Grants with Industry

7.1 Openflexo

We participate to the creation of the Openflexo, a social and solidarity economy company dedicated to cooperation for the development and use of the Openflexo modeling toolkit [5,4]. This company was created with 3 persons and develops and animates a network of 10 cooperators (other companies or individuals).

7.2 Projet Recherche Innovante Thalès

We participate with the Openflexo SCIC to the development of a modeling editor prototype.
8 Other Grants and Activities

8.1 International Collaborations

- Vincent Englebert, from Université Catholique de Namur.

- Siegfried Rouvrais is the French representative at CDIO collaborator’s international meetings (e.g. Stanford 2011, QUT-Brisbane 2012, Aarhus 2013, MIT/Harvard 2013). The CDIO INITIATIVE (acronym for Conceive – Design – Implement – Operate) is a structured educational framework for preparing the next generation of engineers. The framework provides students with an education stressing engineering fundamentals set in the context of Conceiving & Designing & Implementing & Operating real-world systems and products. Throughout the world, now more than 100 recognized institutions have adopted CDIO as the framework of their curricular and programme design, planning, supported by defined verification and validation processes.

- The PhD thesis of HUYNH Ngoc Tho is a grant from the Vietnam-French cooperation.

8.2 National Collaborations

- The PASS team has a contract with Région Bretagne for the funding of S. Martinez’s PhD (since September 2012). This thesis is co-advised by Jérémy Buisson of ArchWare Team of the University of Vannes / IRISA.

9 Dissemination

9.1 Involvement in the Scientific Community

- Antoine Beugnard has been a member of the programming committee of CAL and CIEL 2013. He was also a member in several doctoral committees: Marie Ludwig on “Autonomie et reconfiguration des systèmes de systèmes tactiques”, Mounira Kezadri on “Assistance à la validation et vérification de systèmes critiques : Ontologies et Intégration de composants”, Viet-Hoa Nguyen on “Une méthode fondée sur les modèles pour gérer les propriétés temporelles des systèmes à composants logiciels” and Stéphane Creff on “Une modélisation de la variabilité multidimensionnelle pour une évolution incrémentale des lignes de produits”.

9.2 Teaching

- Antoine Beugnard teaches software engineering, modeling and object programming. He teaches these courses at License and at Master level. He is the coordinator of the Brittany research master (Master Recherche en Informatique de l’Université de Rennes 1) at Télécom Bretagne.

- Fabien Dagnat is the head of the computer science speciality at Telecom Bretagne. He teaches advanced programming (programming languages, functional programming, compilation), formal modeling (for concurrency mainly) and object oriented programming to
master students of Telecom Bretagne. He teaches also a formal modeling course in the core of the brittany research master.

- Julien Mallet teaches software engineering, modeling, object oriented programming and computer security at License and at Master level. He is also involved in project-based learning.

- Siegfried Rouvrais is involved in a broad range of courses relating to computer science and software engineering both at B.Sc. and M.Sc. levels, as well as for adult professionals (e.g., Requirements, Software Architecture, Programming Languages, UML, Business Process Design and Modelling, Workflow, Enterprise Information Systems). In connection with these theoretical and practical courses, he investigates problem- and project-based learning styles aiming at favoring students’ autonomy with a focus on process and reflectivity. Since 2003, as educational program designer, he has been particularly involved in integrating such models into Telecom Bretagne curricula and vocational trainings, with a clear emphasis on student’s competency development, multidisciplinarity, and industry connections. Since several years, he trains faculty members and offer consultations for improving teaching and curriculum design (PjBL, Active Pedagogy, Reflectivity, Constructive Alignment, Programme Design, etc.).

- Maria-Teresa Segarra is the head of the computer science domain at Telecom Bretagne. She teaches advanced programming paradigms (component and service oriented), object-oriented programming and design and databases at License and Master levels.

10 Bibliography

Major publications by the team in recent years


**Doctoral dissertations and “Habilitation” theses**


**Articles in referred journals and book chapters**


Publications in Conferences and Workshops


