AGERE! at SPLASH 2012

2nd International Workshop on Programming based on Actors, Agents, and Decentralized Control

Workshop held at ACM SPLASH 2012
21-22 October 2012
Tucson, Arizona, USA

PROGRAMME and ABSTRACTS

The pre-proceedings are available on the web site at:
`http://agere2012.apice.unibo.it`

AGERE! is an ACM SIGPLAN workshop,
sponsored by Typesafe, Inc.

[11] M. Resnick. *Turtles, Termites and Traffic Jams. Explorations in Massively Parallel Microworlds.* MIT Press, 1994.

[12] A. Ricci and A. Santi. Agent-oriented computing: Agents as a paradigm for computer programming and software development. In *Proc. of Future Computing '11*, Rome, Italy, 2011.

[13] Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, 1993.

[14] H. Sutter and J. Larus. Software and the concurrency revolution. *ACM Queue: Tomorrow's Computing Today*, 3(7):54–62, Sept. 2005.

[15] A. Yonezawa and M. Tokoro. *Object-oriented concurrent programming.* MIT Press series in computer systems. MIT Press, 1987.

Bibliography

[1] *SPLASH '11 Workshops: Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11 & VMIL'11*, New York, NY, USA, 2011. ACM.

[2] G. Agha. Concurrent object-oriented programming. *Commun. ACM*, 33:125–141, September 1990.

[3] G. A. Agha, I. A. Mason, S. F. Smith, and C. L. Talcott. A foundation for actor computation. *J. Funct. Program.*, 7(1):1–72, Jan. 1997.

[4] A. Basu, M. Bozga, and J. Sifakis. Modeling heterogeneous real-time components in bip. In *Proceedings of the Fourth IEEE International Conference on Software Engineering and Formal Methods*, SEFM '06, pages 3–12, Washington, DC, USA, 2006. IEEE Computer Society.

[5] R. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors. *Multi-Agent Programming Languages, Platforms and Applications - Volume 1*. Springer, 2005.

[6] R. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors. *Multi-Agent Programming Languages, Platforms and Applications - Volume 2*. Springer, 2009.

[7] R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni. Special issue on multi-agent programming. *Autonomous Agents and Multi-Agent Systems*, 23 (2), 2011.

[8] D. Harel, A. Marron, and G. Weiss. Behavioral programming. *Commun. ACM*, 55(7):90–100, July 2012.

[9] D. Harel and A. Pnueli. *On the development of reactive systems*, pages 477–498. Springer-Verlag New York, Inc., New York, NY, USA, 1985.

[10] J. J. Odell. Objects and agents compared. *Journal of Object Technology*, 1(1):41–53, 2002.

4.13 "Timed-Rebeca Schedulability and Deadlock-Freedom Analysis Using Floating-Time Transition System" by E. Khamespanah, Z. S. Kaviani, R. Khosravi, M. Sirjani and M-J. Izadi

Timed-Rebeca is an actor-based modeling language for modeling real-time reactive systems. Its high-level constructs make it more suitable for using it by software practitioners compared to timedautomata based alternatives. Currently, the verification of TimedRebeca models is done by converting into timed-automata and using UPPAAL toolset to analyze the model. However, state space explosion and time consumption are the major limitations of using the back-end timed automata model for verification. In this paper, we propose a new approach for direct schedulability checking and deadlock freedom verification of Timed-Rebeca models. The new approach exploits the key feature of Timed-Rebeca, which is encapsulation of concurrent elements. In the proposed method, each state stores the local time of each actor separately, avoiding the need for a global time in the state. This significantly decreases the size of the state space. We prove the bisimilarity of the generated transition system (called floating-time transition system) and the state space generated based on Timed-Rebeca semantics. Also, we provide experimental results showing that the new approach mitigates the state space explosion problem of the former method and allows model-checking of larger problems.

4.14 "Actor Idioms" by D. Schumacher

Actor systems are driven by asynchronous message reception events. Taking full advantage of the Actor Model requires recognizing relevant patterns of actor interaction. We describe several idioms here, in hopes of beginning to build a catalog of useful interactions. Some idioms simply implement already-familiar mechanisms, in terms of actors. Others illustrate novel strategies that can only be realized with asynchronous actor messaging. All provide new perspective on the process of computation.

Contents

latency. Based on the domainspecific constraints, PARTEs design relies on a combination of lock-free data structures, safe memory management techniques, and message passing between Rete nodes. In our benchmarks, we measured scalability up to 8 cores, outperforming highly optimized sequential implementations.

4.12 "A Relational Trace Logic for Simple Hierarchical Actor-Based Component Systems" by I. W. Kurnia and A. Poetzsch-Heffter

We present a logic for proving functional properties of concurrent component-based systems. A component is either a single actor or a group of dynamically created actors. The component hierarchy is based on the actor creation tree. The actors work concurrently and communicate asynchronously. Each actor is an instance of an actor class. An actor class determines the behavior of its instances. We assume that specifications of the behavior of the actor classes are available. The presented logic allows deriving properties of larger components from specifications of smaller components in a hierarchical manner. The behavior of components is expressed in terms of traces where a trace is a sequence of events. A component specification relates traces of input events to traces of output events. Generalizing Hoare-like logics from states to traces and from statements to components, we write p C q to mean that if an input trace satisfies p, component C produces output traces satisfying q; that is, p and q are assertions over traces. Such specifications are partial in that they only specify the reaction of C to input traces satisfying p. This paper develops the trace semantics and specification technique for actor-based component systems, presents important proof rules, proves soundness of the rules, and illustrates the interplay between the trace semantics, the specification technique and the proof rules by an example derived from an industrial Erlang case study.

multiparty interactions with observation. We show that this model provides a natural encoding for priorities and moreover, can be used as an intermediate step towards provably correct and optimized distributed implementations.

4.10 "Distributed Priority Synthesis using Knowledge" by C.-H. Cheng, R. Yan, H. Ruess and S. Bensalem

For distributed computing, orchestrations along predefined communication paths are used to obtain agreement between system components on the next chosen transition. Although the communication overhead can be high, it can be efficiently reduced by the introduction of knowledge, which provides each local component imperfect view on the global state during run-time. In this paper, given a safety criterion, we formulate the problem how to automatically orchestrate components in a system using stateless precedences between actions under the assist of statically computed knowledge. If the system is diagnosed as unsafe, the use of knowledge can be integrated in the synthesis process to enlarge the set of legal fixing candidates. These new solution candidates may disrespect predefined communication paths but their defined priorities are still guaranteed to be deployable.

4.11 "Parallel Gesture Recognition with Soft Real-Time Guarantees" by T. Renaux, L. Hoste, S. Marr and W. De Meuter

Applying imperative programming techniques to process event streams, like those generated by multi-touch devices and full-body motion detection, has significant engineering drawbacks. Declarative approaches solve these problems but have not been able to scale on multicore systems while providing guaranteed response times.We propose PARTE, a parallel scalable complex event processing engine which allows a declarative definition of patterns and provides soft real-time guarantees for their recognition. It extends the state-saving Rete algorithm and maps the event matching onto a graph of actor nodes. Using a tiered event matching model, PARTE provides upper bounds on the detection

1 Introduction

ago, agis, egi, actum, **agere**
latin verb meaning to act, to lead, to do,
common root for actors and agents

"The Free Lunch is Over" also for Abstractions

The fundamental turn of software into concurrency, interactivity, and distribution is not only a matter of performance, but also design and abstraction. *The free lunch is over* [14] calls for devising new programming paradigms – possibly as evolution of existing ones — that would allow for natural ways of thinking about, designing, developing, executing, debugging and profiling systems that exhibit different degrees of concurrency, autonomy, decentralization of control, and physical distribution. Almost any application today requires the programming of software components that actively –proactively and reactively –carry out multiple tasks, react to various kinds of events, and communicate with each other. Relevant research questions include: how to properly program these entities and systems of entities, what kinds of programming abstractions can help in systematically structuring complex reactive and proactive behaviors, and what kinds of programming abstractions can be effective in organizing applications as ensembles of relatively autonomous entities working together.

Actors, Agents and Abstractions for Decentralized Control

Given this premise, in SPLASH 2011 the AGERE! workshop [1] was proposed for the first time to investigate the definition of proper levels of abstraction, programming languages, and platforms to support and promote a decentralized mindset [11] in systems development. To this end, *agents* (and multi-agent systems) and *actors* were taken as a starting point, as two main broad families of concepts described in the literature. These abstractions and programming

tools explicitly promote such a decentralized-control mindset from different facets, depending on the context in which they are discussed, e.g., concurrent programming or distributed artificial intelligence.

Actors [3] and object-oriented concurrent programming [15, 2] couple object-oriented programming with concurrency, providing a clean and powerful computation model which is nowadays increasingly adopted in mainstream languages, frameworks and libraries. Agents and agent-oriented programming [5, 6, 7, 10, 13, 12] provide a rich abstraction layer on top of actors and objects. This approach aims at easing programming of concurrent/distributed systems conceived as societies of autonomous and proactive task-oriented individuals interacting in a shared environment.

The wave of interest on concurrency and distribution in mainstream programming has been clearly witnessed also through the good number of contributions accepted to OOPSLA and OnWard! in SPLASH 2011 (and in other recent editions) that addresses those same issues. However, the main focus in those contributions (including invited talks and panels) so far has been mainly on issues related to performance, and *mechanisms for extending mainstream paradigms* to effectively exploit the power of e.g. multi-core and many-core architectures. While acknowledging the importance of those objectives, at the same time we argue for the importance of strengthening the research on new paradigms aiming *first* at improving the conceptual modeling and the level of abstraction used to design and program such complex software systems.

With that main objective in mind, AGERE! is organized in SPLASH 2012 to promote the investigation of the features that would make agent-oriented and actor-oriented programming languages effective and general-purpose in developing software systems as an evolution of OOP. Besides actors and agents, the workshop is meant, more generally, to serve as a venue for all programming approaches and paradigms investigating how to effectively specify and structure control when programming reactive systems [9, 8, 4] providing new abstractions for dealing, e.g., with management of asynchronous events and the efficient execution of concurrent activities.

nodes/agents leaving the system. We demonstrate the approach with examples taken from sensor networks, and show some experimental results on the QuestMonitor platform.

4.8 "A Decentralized Approach for Programming Interactive Applications with JavaScript and Blockly" by A. Marron, G. Weiss and G. Wiener

We present a decentralized-control methodology and a toolset for developing interactive user interfaces. We focus on the common case of developing the client side of Web applications. Our approach is to combine visual programming using Google Blockly with a single-threaded implementation of behavioral programming in JavaScript. We show how the behavioral programming principles can be implemented with minimal programming resources, i.e., with a singlethreaded environment using coroutines. We give initial, yet full, examples of how behavioral programming is instrumental in addressing common issues in this application domain, e.g., that it facilitates separation of graphical representation from logic and handling of complex inter-object scenarios. The implementation in JavaScript and Blockly (separately and together) expands the availability of behavioral programming capabilities, previously implemented in LSC, Java, Erlang and C++, to audiences with different skill-sets and design approaches.

4.9 "Optimized Distributed Implementation of Multiparty Interactions with Observation" by S. Bensalem, M. Bozga, J. Quilbeuf and J. Sifakis

Using high level coordination primitives allows enhanced expressiveness of component-based frameworks to cope with the inherent complexity of present-day systems designs. Nonetheless, their distributed implementation raises multiple issues, regarding both the correctness and the runtime performance of the final implementation. We propose a novel approach for distributed implementation of multiparty interactions subject to scheduling constraints expressed by priorities. We rely on new composition operators and semantics that combine

oriented concurrent programming or by the actor model, being them natively based on the reactivity principle only. In this paper we tackle the problem with agent-oriented programming, using an agent-oriented programming language called simpAL.

4.6 "Empirical Software Engineering for Agent Programming" by M. B. van Riemsdijk

Empirical software engineering is a branch of software engineering in which empirical methods are used to evaluate and develop tools, languages and techniques. In this position paper we argue for the use of empirical methods to advance the area of agent programming. Through that we will complement the solid theoretical foundations of the field with a thorough understanding of how our languages and platforms are used in practice, what the main problems and effective solutions are, and how to improve our technology based on empirical findings. Ultimately, this will pave the way for establishing multi-agent systems as a mature and recognized software engineering paradigm with clearly identified advantages and application domains.

4.7 "Messages with Implicit Destinations as Mobile Agents" by A. Ahmad-Kassem, S. Grumbach and S. Ubéda

Applications running over decentralized systems, distribute their computation on nodes/agents, which exchange data and services through messages. In many cases, the provenance of the data or service is not relevant, and applications can be optimized by choosing the most efficient solution to obtain them. We introduce a framework which allows messages with intensional destination, which can be seen as restricted mobile agents, specifying the desired service but not the exact node that carries it, leaving to the system the task of evaluating the extensional destination, that is an explicit address for that service. The intensional destinations are defined using queries that are evaluated by other agents while routing. We introduce the Questlog language, which allows to reformulate queries, and express complex strategies to pull distributed data. In addition, intensional addresses offer persistency to dynamic systems with

Theory and Practice of Programming with Actors, Agents and Decentralized Control Abstractions

All stages of software development are considered interesting for the workshop, including requirements, modeling, prototyping, design, implementation, testing, and any other means of producing running software based on actors and agents as first-class abstractions. The scope of the conference includes aspects that concern both the theory and the practice of design and programming using such paradigms, so as to bring together researchers working on the models, languages and technologies, as well as the practitioners using such technologies to develop real-world systems and applications.

Finally, the overall perspective of the workshop is what distinguishes this event from related venues (e.g. about agents) organized in different contexts (e.g. AI) with the intent to hopefully impact mainstream programming paradigms and software development. Another purpose of the workshop is to serve as a forum for collecting, discussing, and confronting related research work that typically appears in different communities in the context of (distributed) artificial intelligence, distributed computing, computer programming, and software engineering.

Acknowledgment

The organizing committee would like to thank all program committee members, authors and participants. Thank you to ACM and SPLASH organizers, and to Typesafe Inc., for their support. We look forward to a productive workshop.

4.4 "Adding Distribution and Fault Tolerance to Jason" by Á. Fernández Díaz, C. Benac Earle and L.-A. Fredlund

In this paper we describe an extension of the multiagent system programming language Jason with constructs for distribution and fault tolerance. The standard Java-based Jason implementation already does provide a distribution mechanism, which is implemented using the JADE library, but to use it effectively some Java programming is often required. Moreover, there is no support for fault tolerance. In contrast, this paper develops constructs for distribution and fault tolerance wholly integrated in Jason, permitting the Jason programmer to implement complex distributed systems entirely in Jason itself. The fault tolerance techniques implemented allow the agents to detect, and hence react accordingly, when other agents have stopped working for some reason (e.g., due to a software or a hardware failure) or cannot be reached due to a communication link failure. The introduction of distribution and fault tolerance in Jason represents a step forward towards the coherent integration of successful distributed software techniques into the agent based software paradigm. The proposed extension to Jason has been implemented in eJason, an Erlang-based implementation of Jason. In fact, in this work we essentially import the distribution and fault tolerance mechanisms from the Erlang programming language into Jason, a task which requires the adaptation of the basic primitives due to the difference between a process based functional programming language (Erlang) and a language for programming BDI (Belief-Desire-Intention) agent based systems (Jason).

4.5 "Programming Abstractions for Integrating Autonomous and Reactive Behaviors: An Agent-Oriented Approach" by A. Ricci and A. Santi

The integration of autonomous and reactive behavior is a relevant problem in the context of concurrent programming, related to the integration of thread-based and event-driven programming. From a programming paradigm perspective, the problem can not be easily solved by approaches based on object-

Given an Erlang module and a set of properties, Soter first extracts an abstract (approximate but sound) model in the form of an actor communicating system (ACS), and then checks if the properties are satisfied using a Petri net coverability checker, BFC. To our knowledge, Soter is the first fully-automatic, infinite-state model checker for a large fragment of Erlang. We find that in practice our abstraction technique is accurate enough to verify an interesting range of safety properties such as mutual-exclusion and boundedness of mailboxes. Though the ACS coverability problem is EXPSPACE-complete, Soter can analyse these problems surprisingly efficiently.

4.3 "Leveraging Actors for Privacy Compliance" by J. von Ronne

Many organizations store and process personal information about the individuals with whom they interact. Because incorrect handling of this information can be harmful to those individuals, this information is often regulated by privacy policies. Although noncompliance can be costly, determining whether an organizations systems and processes actually follow these policies is challenging. It is our position, however, that such information systems could be formally verified if it is specified, designed, and implemented according to a methodology that prioritizes verifiability of privacy properties. This paper describes one such approach that leverages an actor-based architectural style, formal specifications of personal information that is allowed and required to be communicated, and a domain-specific actor-based language. Specifications at the system-, componentActor-level are written using a first-order temporal logic. We propose that the software implementation be mechanically-checked against individual actor specifications using abstract interpretation. Whereas, consistency between the different specification levels and would be checked using model checking. By restricting our attention to programs using a specific actor-based style and implementation technology, we can make progress towards the very challenging problem of rigorously verifying program implementations against complex privacy regulations.

2 Committee

2.1 Program Committee

Gul Agha, University of Illinois at Urbana-Champaign, USA
Joe Armstrong, SICS / Ericsson, Sweden
Saddek Bensalem, Verimag, France
Rafael H. Bordini, FACINPUCRS, Brazil
Gilad Braha, Google, USA
Rem Collier, UCD, Dublin
Tom Van Cutsem, Vrije Universiteit, Brussel, Belgium
Amal El Fallah Seghrouchni, LIP6 Univ. P and M. Curie, Paris, France
Jurgen Dix, Technical University of Clausthal, Germany
Philipp Haller, Typesafe, Switzerland
Tom Holvoet, Dept. Computer Science K.U.Leuven, Belgium
Einar Broch Johnsen, University of Oslo, Norway
Hillel Kugler, Microsoft, USA
Assaf Marron, Weizmann Institute of Science, Israel
Mark Miller, Google, USA
Olaf Owe, University of Oslo, Norway
Jens Palsberg, UCLA, Los Angeles, USA
Ravi Pandya, Microsoft, USA
Arnd Poetzsch-Heffter, University of Kaiserslautern, Germany
Alessandro Ricci, University of Bologna, Italy
Birna van Riemsdijk, Delft University of Technology, The Netherlands
Giovanni Rimassa, Whitestein Technologies, Switzerland
Munindar Singh, North Carolina State University, USA
Marjan Sirjani, Reykjavik University, Iceland
Gera Weiss, Ben Gurion University, Israel
Guy Wiener, HP, Israel
Akinori Yonezawa, University of Tokyo, Japan

2.2 Organizing Committee & PC Chairs

Gul Agha, University of Illinois at Urbana-Champaign, USA
Rafael H. Bordini, FACIN–PUCRS, Brazil
Assaf Marron, Weizmann Institute of Science, Israel
Alessandro Ricci, University of Bologna, Italy

4 Abstracts

4.1 "Domains: Safe sharing among actors" by J. De Koster, T. Van Cutsem and T. D'Hondt

The actor model has already proven itself as an interesting concurrency model that avoids issues such as deadlocks and race conditions by construction, and thus facilitates concurrent programming. While it has mainly been used in a distributed context it is certainly equally useful for modeling interactive components in a concurrent setting. In component based software, the actor model lends itself to naturally dividing the components over different actors and using message passing concurrency for implementing the interactivity between these components. The tradeoff is that the actor model sacrifices expressiveness and efficiency especially with respect to parallel access to shared state. This paper gives an overview of the disadvantages of the actor model in the case of shared state and then formulates an extension of the actor model to solve these issues. Our solution proposes domains and synchronization views to solve the issues without compromising on the semantic properties of the actor model. Thus, the resulting concurrency model maintains deadlock-freedom and avoids low-level race conditions.

4.2 "Soter: An Automatic Safety Verifier for Erlang" by E. D'Osualdo, J. Kochems and L. Ong

This paper presents Soter, a fully-automatic program analyser and verifier for Erlang modules. The fragment of Erlang accepted by Soter includes the higher-order functional constructs and all the key features of actor concurrency, namely, dynamic and possibly unbounded spawning of processes and asynchronous message passing. Soter uses a combination of static analysis and infinite-state model checking to verify safety properties specified by the user.

3 Programme

3.1 First Day (Oct 21, Sunday)

- 8:30 9:00 Welcome
  - Introduction to the AGERE! workshop: Motivation, Objectives, Agenda.
- 9:00 10:00 Invited talk

  On the integration of the actor model in mainstream technologies The Scala perspective **by Philipp Haller, Typesafe**
- 10:00 10:30 Coffee Break
- 10:30 12:00 Research Papers session I Actors
  - *Domains: Safe sharing among actors*
    Joeri De Koster, Tom Van Cutsem and Theo D'Hondt – Vrije Universiteit (Belgium)
  - *Soter: An Automatic Safety Verifier for Erlang*
    Emanuele D'Osualdo, Jonathan Kochems and Luke Ong – Oxford University (United Kingdom)
  - *Leveraging Actors for Privacy Compliance*
    Jeffery Von Ronne – The University of Texas at San Antonio (United States)
- 12:00 13:30 Lunch
- 13:30 14:30 Invited talk

  20 years of Agent-Oriented Programming in Distributed AI: History and Outlook **by Birna van Riemsdijk Delft University of Technology, The Netherlands**

- 14:30 15:00 Research Paper Session IIa Agent-Oriented Programming
  - *Adding Distribution and Fault Tolerance to Jason*
    Álvaro Fernández Díaz, Clara Benac Earle and Lars-Ake Fredlund – Universidad Politcnica de Madrid (Spain)
- 15:00 15:30 Coffee Break
- 15:30 17:00 Research Paper Session IIb Agent-Oriented Programming
  - *Programming Abstractions for Integrating Autonomous and Reactive Behavior: An Agent-Oriented Approach*
    Alessandro Ricci, Andrea Santi – University of Bologna (Italy)
  - *Empirical Software Engineering for Agent Programming*
    Birna Van Riemsdijk – TU Delft (The Netherlands)
  - *Messages with Implicit Destinations as Mobile Agents*
    Ahmad Ahmad-Kassem, Stphane Grumbach and Stphane Ubda – INRIA INSA Lyon, INRIA (France)

3.2 Second Day (Oct 22, Monday)

- 8:30 9:00 Welcome
  - Brief summary of the first day and agenda of the second day. Preparation for the panel discussion.
- 9:00 10:00 Invited Talk

  Agents, Concurrent Objects, and High Performance Computing by Akinori Yonezawa University of Tokyo, Japan
- 10:00 10:30 Coffee Break
- 10:30 12:00 Research Paper Session III Decentralized control programming with Behavioral Abstractions
- Introduction by Assaf Marron
  - *A Decentralized Approach for Programming Interactive Applications with JavaScript and Blockly*
    Assaf Marron, Gera Weiss and Guy Wiener – Weizmann Institue of Science, Ben Gurion University, HP Labs (Israel)

  - *Optimized Distributed Implementation of Multiparty Interactions with Observation*
    Saddek Bensalem, Marius Bozga, Jean Quilbeuf and Joseph Sifakis – VERIMAG, VERIMAG/CNRS (France)
  - *Distributed Priority Synthesis using Knowledge*
    Chih-Hong Cheng, Rongjie Yan, Harald Ruess and Saddek Bensalem – Fortiss (Germany), State Key Laboratory of Computer Science Institute of Software (China), VERIMAG (France)

- 12:00 13:30 Lunch time

- 13:30 15:00 Research Paper Session IV Actors
  - *Parallel Gesture Recognition with Soft Real-Time Guarantees*
    Thierry Renaux, Lode Hoste, Stefan Marr and Wolfgang De Meuter – Vrije Universiteit (Belgium)
  - *A Relational Trace Logic for Simple Hierarchical Actor-Based Component Systems*
    Ilham W. Kurnia and Arnd Poetzsch-Heffter – University of Kaiserslautern (Germany)
  - *Timed-Rebeca Schedulability and Deadlock-Freedom Analysis Using Floating-Time Transition System*
    Ehsan Khamespanah, Zeynab Sabahi Kaviani, Ramtin Khosravi, Marjan Sirjani and Mohammad-Javad Izadi – Tehran University (Iran), Reykjavik University (Iceland)

- 15:00 15:30 Coffee Break

- 15:30 17:00 Research Paper V and Discussion Session
  - *Actor Idioms*
    Dale Schumacher – United States
  - Research directions for Agent, Actor and Decentralized Control: panel and open discussion

- 17:00 SPLASH Poster session with one poster about AGERE! summary & contributions