Alexander Bolotov

AUTOGRID: Temporal Modelling of Intelligent Grids

A. Bolotov, V.Getov, L.Henrio, A.Basukoski

Harrow School of Computer Science,

University of Westminster, UK

M.Urbanski University of Poznan, Poland

Overview

- Philosophy
- AUTOGRID Architecture
  - Formal Specifications
  - Dynamism
  - Problem Solving
- Techniques invoked
- Current and future work

CONTEXT

The framework of Task 3.2 Components and Hierarchical Composition:

Definition of a primitive component and its features and the way how components can be hierarchically composed

We aim to build

An environment in which reconfiguration of components in a Grid system can be managed in a safe and optimal way

Building a System - Two approaches

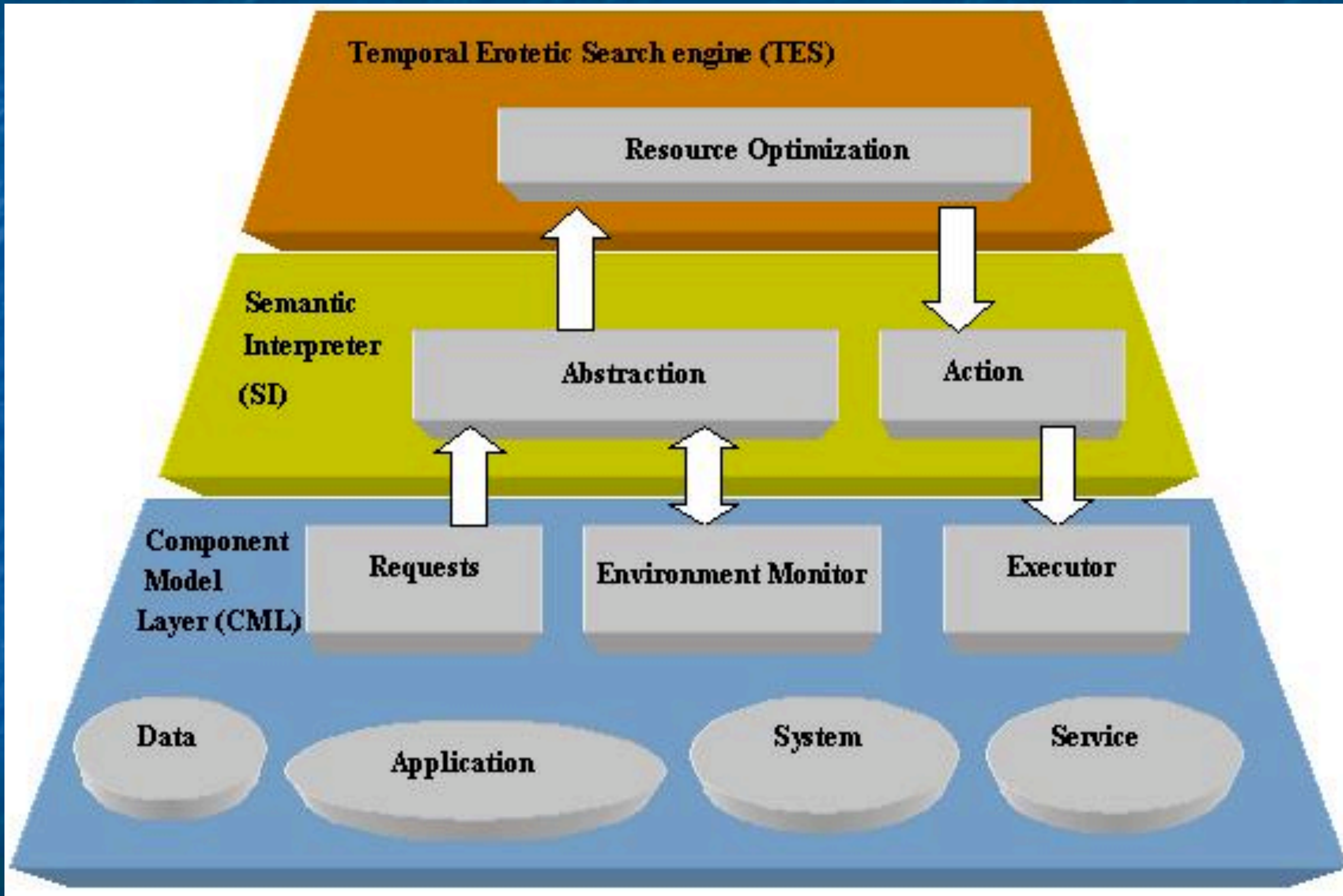
- Two approaches: Exhaustive approach and Generic
- Exhaustive:
  - A system is designed to satisfy every service request from applications
- Generic:
  - A lightweight platform is designed to satisfy only the basic set of service requests enabling its ***reconfiguration*** **and** ***expansion***
  - Identify the basic set of features of the component model
  - Consider any other functions as pluggable components

Intelligency in Grid - Self-Organisation

- Exhaustive approach leads to very high complexity
- Generic approach requires intelligent techniques to manage ***reconfiguration*** **and** ***expansion***
- ***AUTOGRID – an intelligent system which addresses these problems***

University of Westminster

Architecture of AUTOGRID

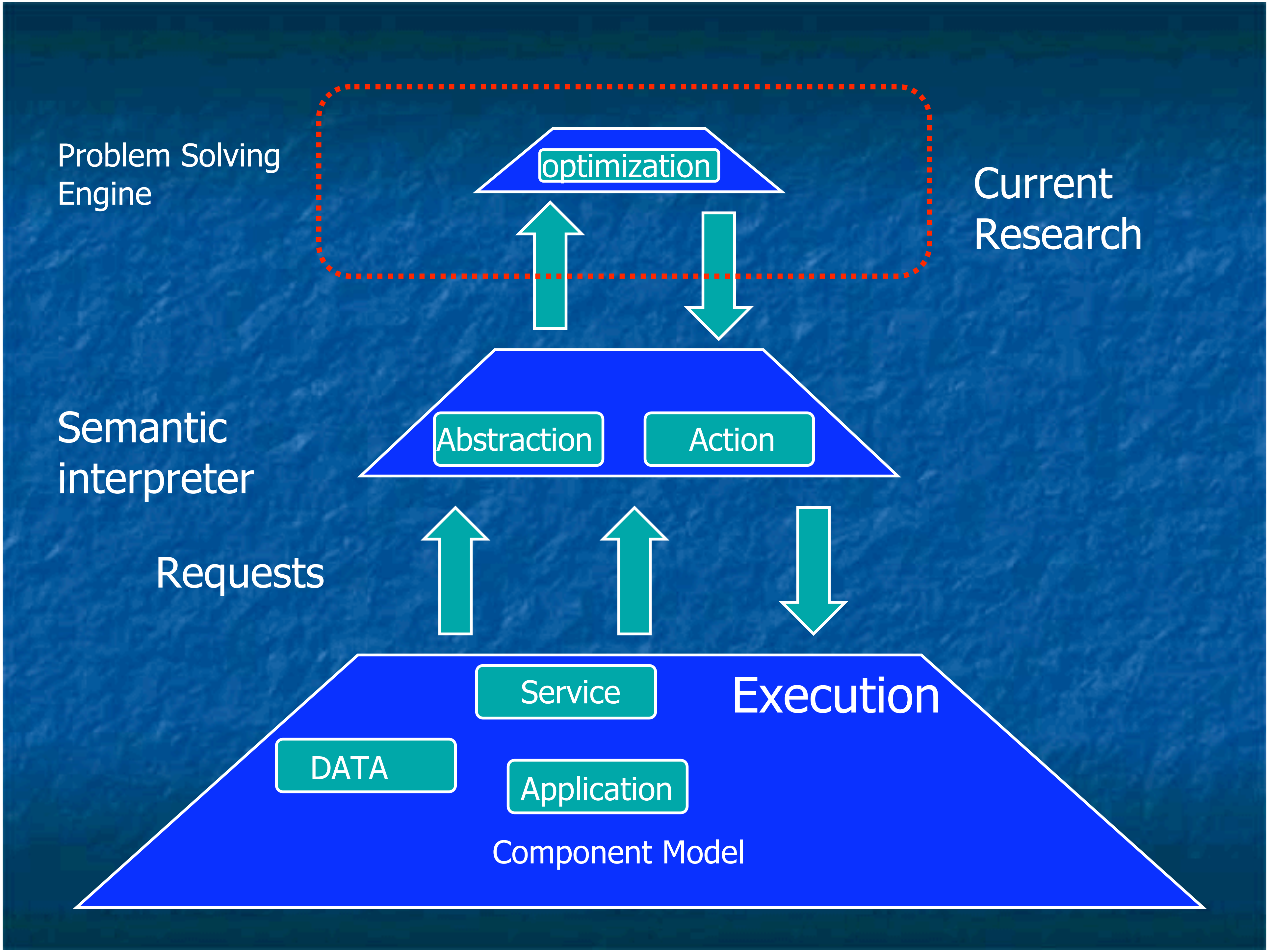


Our Starting Points

- Can view components as AGENTS
- Ideologically similar to a model presented by Jean Bernard Stefani: Component is a Runtime entity
- The overall system design as a multi-layered multi-agent system
- Enables use of agent based formal specification and verification

AUTOGRID Information Flow

- Multi-layered self-organizing generic Grid architecture
- Requests on reconfigurations of components of a component model are automatically generated
- A problem-solving (TESS engine) engine automatically provides optimal solutions
- Provide efficient information flow to enable passing the requests to this engine and the solutions back (Semantic Interpreter)


Problem Solving Engine
optimization
Current Research
Semantic interpreter
Abstraction
Action
Requests
Service
Execution
DATA
Application
Component Model

Various Levels of Specification and Verification

- Component Model Specification - Requests on reconfigurations
- Specification of dynamic system - Framework of Temporal Logic
  - Prove core correctness properties of the underlying systems e.g., verify the behaviour of a system of components
- Temporal Reasoning over Trees
  - Potentially apply well established methods of automated reasoning such as verification and problem solving

High Level Specification and Verification
Temporal Representation and Reasoning

- High-level specification techniques
- → Prove core correctness properties of the underlying systems
- Potentially apply well established methods of automated reasoning such as verification and problem solving
- → e.g., Verify the behaviour of a system of components

Examples of Temporal Logic Specifications

- Invariant: a process **P** is preserved from some point on, i.e. 'always **P** ', i.e. **P becomes an invariant**
- It will be always the case that a property **P** eventually holds
- I want the condition **Q** to eventually happen such that it will be true always until **P** eventually holds
- No two processes **P** and **Q** reach a critical point at the same time.
- A configuration $\mathbf{C_n}$ is accessible from the configuration $\mathbf{C_m}$

Semantic Interpterer

- Requests on reconfigurations of the component model should be interpreted for the problem solving engine
- Assumption: Component = agent
- Allows to invoke agent based specification techniques
- We suggest that this is done in form of abstraction which is then interpreted via automata
- Input from the automata can be translated into the temporal logic specification

Temporal Logic Specification and Verification

- Temporal framework – specific language initially developed for the computation tree logic CTL

  (A. Bolotov, University of Westminster)
- Specific expressive formalism which captures
  - Initial Conditions
  - Transitions of the system
  - Acceptable conditions (eventualities)
  - Interpreted over tree canonical models
- Enables application of deductive techniques such as resolution and extensions to erotetic deductive reasoning

Problem Solving Engine

- Novel problem solving technique based upon the fusion of the frameworks of erotetic and temporal logic.
- Formal representation of questions and corresponding reasoning (A.Wisniewski & M.Urbanski, University of Poznan, Poland).

Current work

- Famous problem in automata based methods – state explosion!
- Investigation of expressiveness of temporal logic specification and tree automata
- Enables use of deductive techniques
- Tree automata = modal mu-calculus
- Can we capture this expressive system in our framwork? If yes then we will enable
- Fixpoint calculations in a dynamic environment

Future Works

- Interpretation of abstraction and requests in terms of automata