A Component-based Approach for Constructing High-confidence Distributed Embedded Systems

Barrett Bryant[1], *Rajeev Raje*[2], *Mikhail Auguston*[3], *Jeff Gray*[1], *Shih-Hsi Liu*[1], *Mihran Tuceryan*[2] *and Andrew Olson*[2]

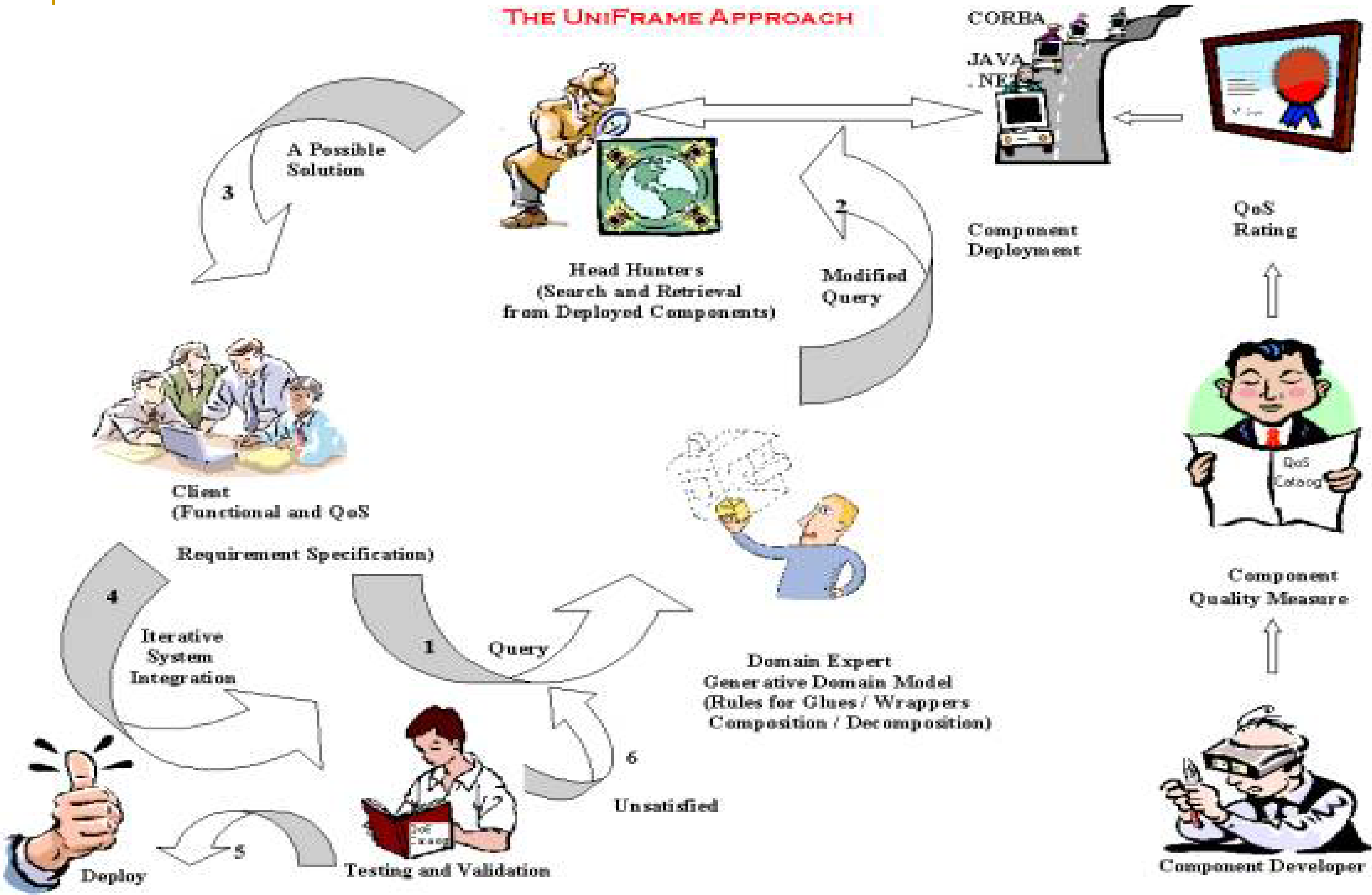
1. University of Alabama at Birmingham
2. Indiana University-Purdue University Indianapolis
3. Naval Postgraduate School

Realizing Distributed Embedded Systems Using Service-Oriented Architectures

- DES as a composition of heterogeneous, independently developed components
- Each component offers services along with associated assurances about them.
- Confidence characteristics incorporated during design, construction, deployment, and composition of these services
- Cost of verification and validation reduced

Research Goals

- Develop service-oriented models for DES which incorporate high-confidence characteristics such as correctness and QoS
- Develop, discover and select components using service-oriented models, so that components and their ensemble exhibit high confidence
- Automate the composition of components to minimize vulnerability arising from handcrafting
- Validate the assembled DES with respect to both functional correctness and QoS


THE UNIFRAME APPROACH
CORBA
JAVA
.NET
A Possible Solution
3
Head Hunters (Search and Retrieval from Deployed Components)
2
Modified Query
Component Deployment
QoS Rating
Client (Functional and QoS Requirement Specification)
QoS Catalog
Component Quality Measure
4
Iterative System Integration
1
Query
Domain Expert Generative Domain Model (Rules for Glues / Wrappers Composition / Decomposition)
6
Unsatisfied
5
Deploy
Testing and Validation
Component Developer

Key Research Issues

- Architecture-based Interoperability
  - Automation, standardization, mappings and tools
- Distributed Resource Discovery
  - Specification, publication, distribution, selection
- Validation of Quality Requirements
  - Vocabulary and associated metrics, composition, monitoring

Main Challenge: Heterogeneity

UniFrame Knowledge Base

- Developed by domain experts for specific application domains
- Describes service-oriented architecture for the application
- Specifies functional and QoS properties of components that make up the architecture
- Discovers and matches components to the requirements
- Automatically generates code for interoperation of components
- Predicts and empirically measures vulnerability properties of the integrated system

Formal Methods

- Language for describing rules for integrating components – Two-Level Grammar (TLG)
- Automated scenario generation from environment models – Attributed Event Grammar (AEG)

Two-Level Grammar

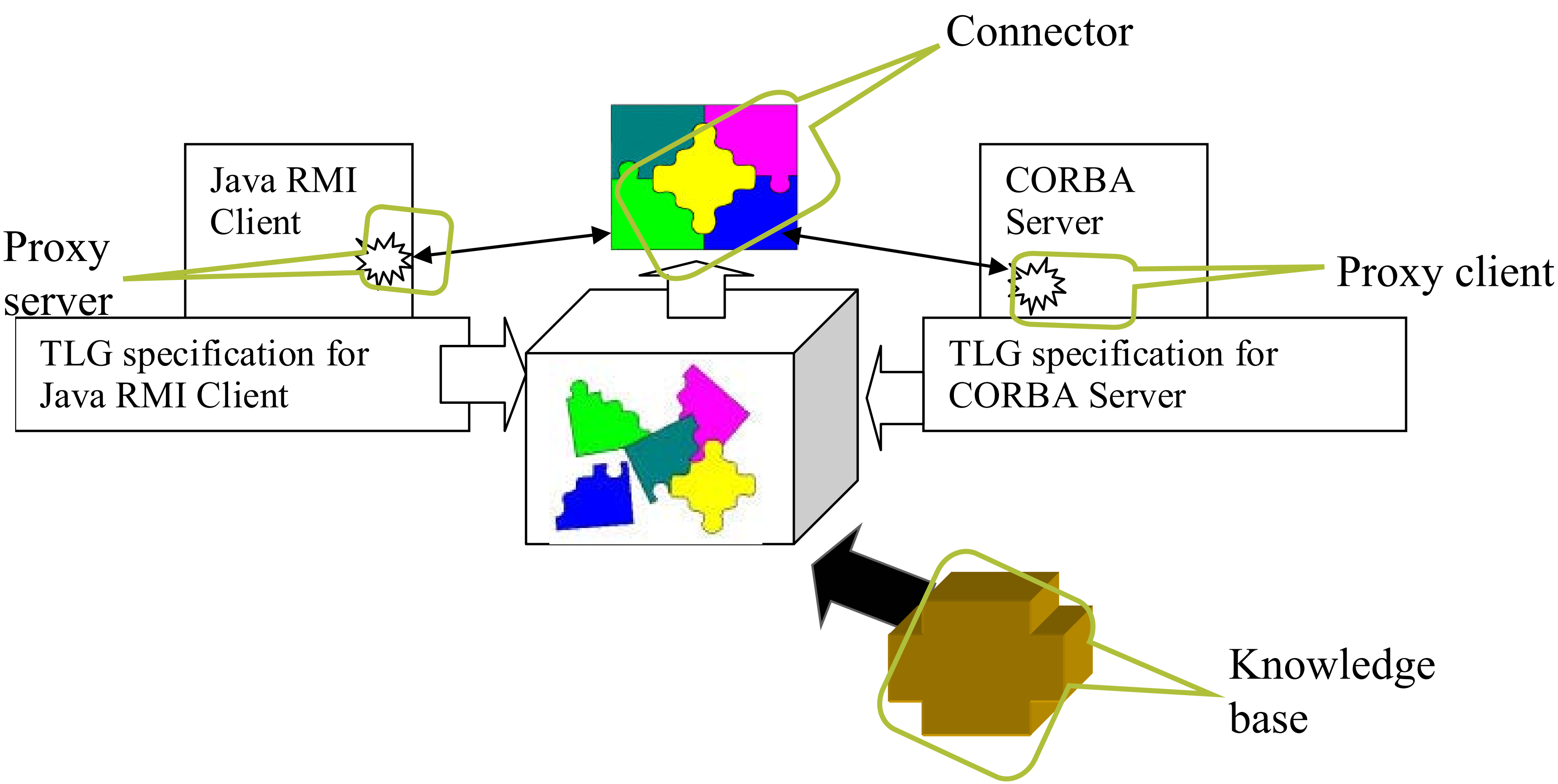
- TLG consists of two context-free grammars corresponding to the set of type domains and the set of logical rules operating on those domains.
- The first level of the grammar, called *meta-rules*, defines the structure of the domain, including the syntactic interfaces of components.
- The second level of the grammar, called *hyper-rules*, defines the rules for composing components, performing static evaluation of QoS constraints, and generation of connector code.

TLG Example

ClientUMM, *ServerUMM* :: *UniframeMetaModel*.
ClientOperations, *ServerOperations* :: {*Interface*}*.

generate *Application* **system**
from *ClientUMM* **and** *ServerUMM* **with** *QoS* **:**
ClientOperations **:=** *ClientUMM* **get operations,**
ServerOperations **:=** *ServerUMM* **get operations,**
OperationMapping **:= map** *ClientOperations* **into**
ServerOperations **using** *Application* **domain,**
ComponentModel **:=**
ServerUMM **get component model,**
generate java code for *OperationMapping*
using *ComponentModel* **with** *QoS*.

TLG Glue/Wrapper Generation



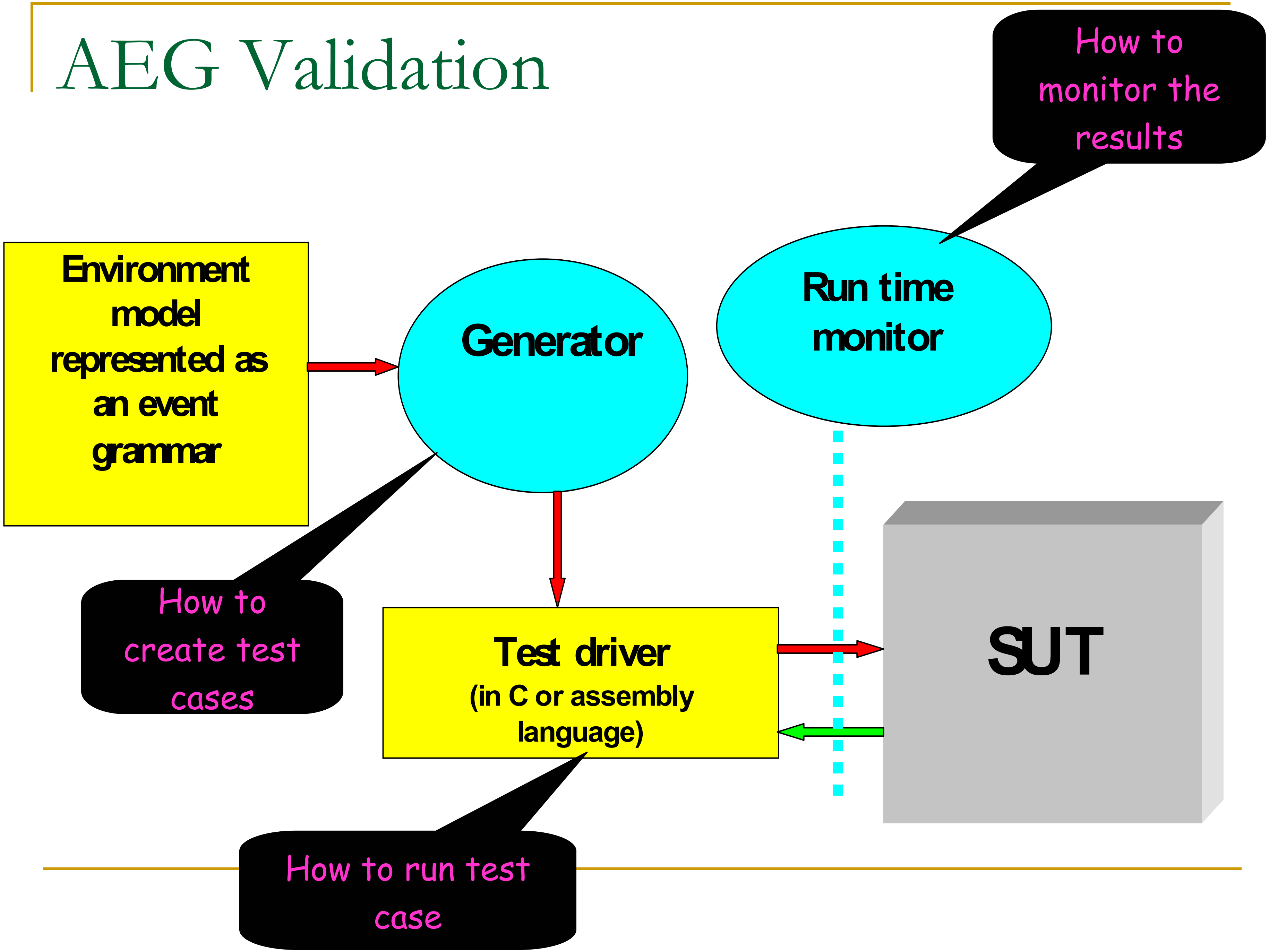
Attributed Event Grammar

- Attributed event grammar (AEG) provides a uniform approach for automatically generating, executing, and analyzing tests.
- Quantitative and qualitative risk assessment can be performed based on statistics gathered during automatic test execution.
- AEG provides automated testing of distributed real-time embedded software systems, based on modeling the environment in which a system will operate.

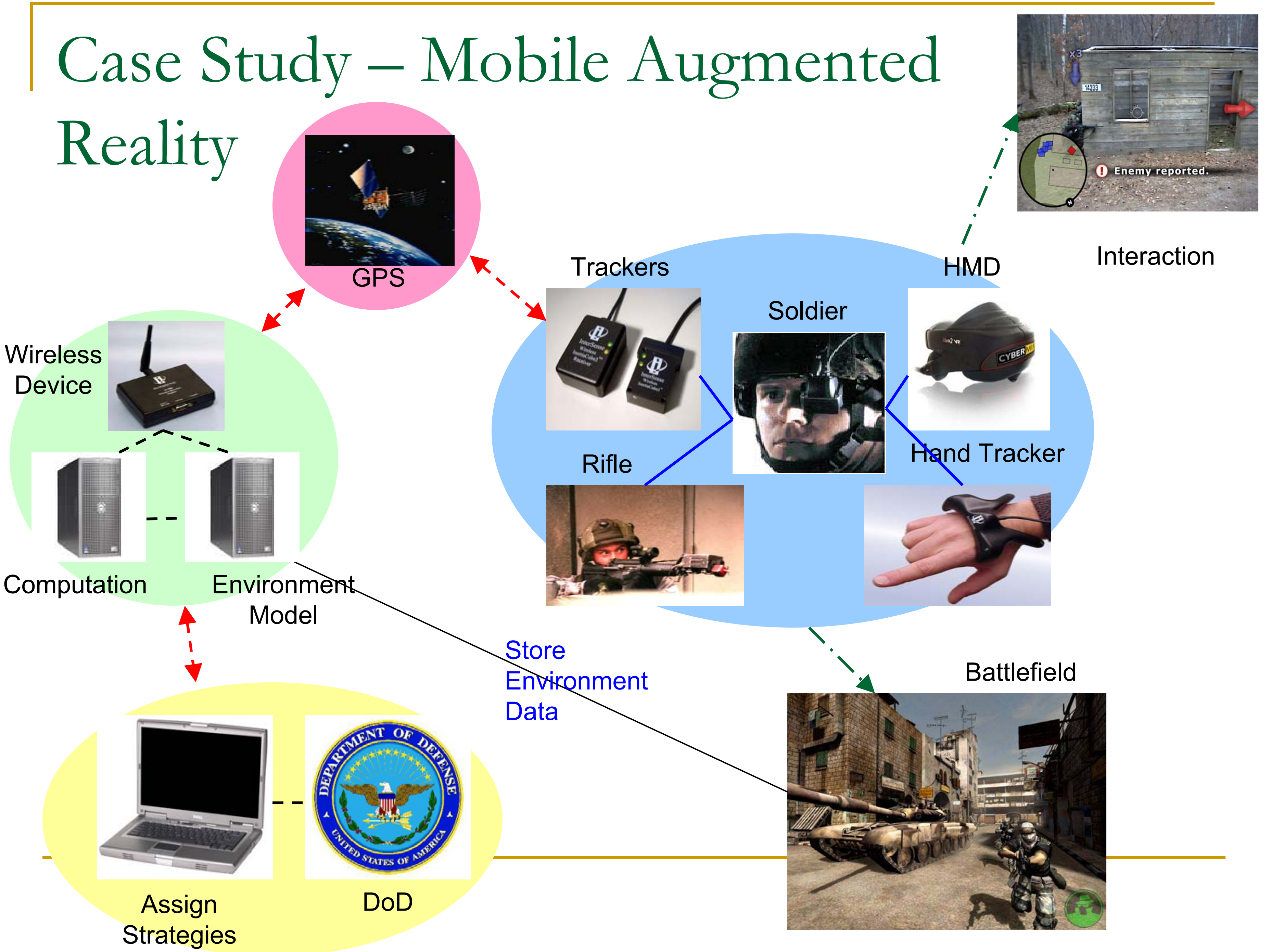
AEG Example

- Shoot ::= Fire
  (p(0.3) Hit /Send_input_to_SUT (Hit . time)/ |
  p(0.7) Miss)
- Large number of Shoot scenarios can be generated.
- Each event trace will satisfy the constraints imposed by the event grammar.

AEG Validation



Case Study – Mobile Augmented Reality



Conclusions

- Development and reuse of existing software components for embedded systems in a manner that fosters high-confidence
- Partially automates the software design and validation process for embedded systems, thereby increasing reliability
- Assists in the development of standards for software component descriptions in embedded domains

Future Work

- Expand case studies to include other domains
- Develop prototype tool suites to further validate framework

Further Information

http://www.cs.iupui.edu/uniFrame