Cloud Resources-Events-Agents Model: Towards TOSCA-Based Applications

Soheil Qanbari[1], Vahid Sebto[2], and Schahram Dustdar[1]

[1] Technical University of Vienna
{qanbari,dustdar}@dsg.tuwien.ac.at
http://dsg.tuwien.ac.at
[2] Baha'i Institute for Higher Education (BIHE)
{vahid.sebto}@bihe.org
http://www.bihe.org

Abstract. The dilemma for domain experts and developers during design time of a cloud application is ensuring the sufficient programming abstractions between them in mapping the business requirements to cloud specifications. Thus, a modeling language is needed to capture and express the business requirements. Resources-Events-Agents (REA) is a well-known business requirement modeling language that decomposes the information system into three constituents with the set of compliant binary collaborations called, *Duality*. This study is a preliminary attempt to employ REA for developing cloud applications. In this study, we define a conceptual mapping between REA model and OASIS Topology and Orchestration Specification for cloud Applications (TOSCA) policies, plans and templates. Based on that, we proceed with the process of building business-driven cloud applications. In support of our model, we implement a cloud REA Modeling tool referred to as CREAM, where business requirements are specified in REA, then corresponding cloud application is composed and built. We describe the underlying mapping strategy as well as the details of our tool in support of the proposed approach.

Keywords: Cloud application, Resources-Events-Agents (REA), TOSCA, Business requirements.

1 Introduction

The cloud abstraction model delivers a shared pool of configurable computing resources (processors, storage, applications, etc.) that can be dynamically and automatically provisioned and released [1]. This elastic delivery of cloud resources improves business agility by enabling the providers to respond faster to the demanding needs of the markets. Firms benefit from this as an enabler in developing adaptive business models built upon cloud applications that meet both business and customer needs. Thus, they can orchestrate processes, (de)allocate resources, (de)provision services and seamlessly adapt to the constantly changing

M. Villari et al. (Eds.) : ESOCC 2014, LNCS 8745, pp. 160–170, 2014.

requirements of their clients. Cloud adaptive business modeling, poses challenges of performing an ongoing assessments to ensure compliance and alignment between business requirements and system specifications.

In architecting cloud applications, the cloud market-leader, Amazon web services (AWS), offers a *CloudFormation*[1] service where we can create a stack to seamlessly provision the collection of resources required by applications. We can deploy CloudFormation's *templates*[2] or create our own templates to describe the AWS resources with associated dependencies or runtime parameters, required to run our applications. The cloud management platform, *OpenStack* provides a service called *Heat*[3] to orchestrate multiple composite cloud applications using the AWS CloudFormation template format, through both an OpenStack-native REST API and a CloudFormation-compatible Query API. The *Heat* engine's main responsibility is to orchestrate the launching of templates and provide events back to the API consumer. On a similar service, the Ubuntu open-source community, provides Ubuntu JuJu[4], a service orchestration management tool where we can define the technical requirements and specifications of our cloud application and proceed with its deployment. Similarly, the openTOSCA[5] provides a container where we can define and run our TOSCA-based cloud application implementation artifacts composed into the cloud Service Archive (CSAR) file which includes the service topology and its implementation plans.

Suffice to say that these initiatives are more focused on capturing technical requirements rather than business models. Such solutions are appropriate for cloud application developers and pose limitations for business developers who know the domain knowledge best but with limited programming skills. There are several well-established business modeling frameworks,including e3-value [3], Resource-Event-Agent (REA) [4] and the Business Modeling Ontology (BMO) [5]. These models allow shorter development cycles and faster time to products and value. However, at the moment, to the best of our knowledge, there is no engagement between the current business modeling frameworks and cloud computing business models. In this paper, we provide this mapping and ultimately, show how effective our tooling is. In summary, our contribution is twofold as follows:

- Analyzing the contemporary business modeling frameworks on which firms base their service identification, specification, and realization strategies.
- The mapping rules between the REA model and the TOSCA model. We implement a tool in support of these compliance rules.

The paper continues with a background in the cloud REA model in section 2 in support of proper positioning of the CREAM tool. Section 3 introduces the REA business modeling framework as an input model. In section 4 TOSCA specifications as an output model are detailed. Section 5 presents the actual contribution of the paper, the conceptual mapping rules together with their supporting

[1] `http://aws.amazon.com/cloudformation/`
[2] `http://aws.amazon.com/cloudformation/aws-cloudformation-templates/`
[3] `https://wiki.openstack.org/wiki/Heat`
[4] `https://juju.ubuntu.com`
[5] `http://www.iaas.uni-stuttgart.de/OpenTOSCA/indexE.php`

facts. Next, the CREAM tool architecture is presented in section 6 and a sample use-case scenario is given to support the efficiency and utilization of our tool. Subsequently, section 7 surveys some scientific related work. Finally, section 8 concludes the paper and presents an outlook on future research directions.

2 Related Work

In relation to our approach, there are some prominent approaches for defining the cloud value chain reference model[7], like an $i*$[6], a goal-oriented social modeling framework for linking business models to their supporting services and process models by Jaap et al[8] and Ramel et al[9]. In their approach, first, the business requirements are modeled with the $i*$ notation and then business services are derived. In the second phase, the identified services are refined according to these requirements using UML activity and class diagrams. On a similar approach, Gailly et al[10] defined a set of business rules to transform the REA meta-model into a UML class diagram with accompanying OCL constraints. Schuster et al[11] leverages model driven development and provide a mapping from REA to UMM. In support of this mapping, Sonnenberg et al[12], developed a domain specific modeling language called REA-DSL. Another more conceptual approach exploiting service science perspective on REA business modeling is introduced by Roelens et al[13]. The authors specify six design criteria to evaluate the ability of REA business model to create service interaction model. Poels et al[14] propose the Resource-Service-System model adapted from REA as a conceptual model for service science that emphasizes the service systems interaction through the exchange of resource for more utilization. To the best of our knowledge, the existing approaches do not address the cloud computing business models as we aim to do by a mapping from REA modeling language to cloud TOSCA model. Next, we explore each of them as an input and an output models of our mapping process.

3 REA – The Input Model

The REA (Resources-Events-Agents) model focuses on the value of business objects exchanged among parties and abstracts away the implementation details of the system to business developers. Figure 1, illustrates the core concepts of REA. Now we delve into the core concepts, their meanings and interdependencies:

♢ **Economic Resource** is a thing that has utility for *Agents*. In fact, users need to deploy, monitor, and utilize the resources. For instance, economic resources can be products, tools, services and humans as well.

♢ **Economic Agent** is a stakeholder or organization capable of having control over economic resources, with an interest in it. Agents deal on resources upon their established service level agreements. Examples of economic agents are consumers, vendors, employees, and third-party enterprises.

[6] `http://www.cs.toronto.edu/km/istar`

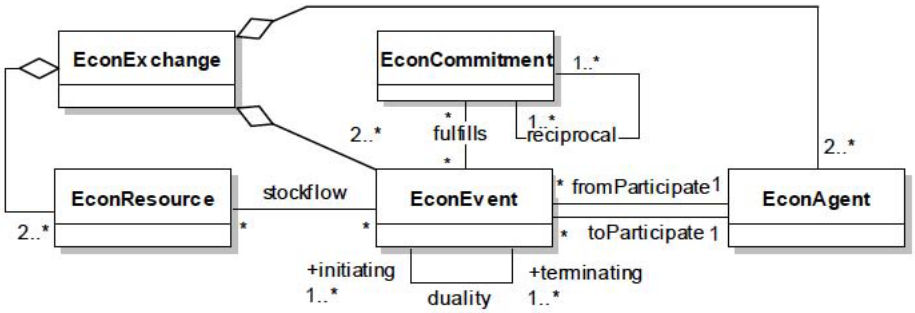

Fig. 1. Excerpt of the REA meta-model and core concepts

♢ **Economic Event** represents either an increment or a decrement in the value of economic resources. Some economic events are demand, supply of resources. Events can be classified into two poles of *Take* and *Give*. At least one *take* event and one *give* event exist for each resource. When the event occurs, the provider loses rights to the resource, and the consumer receives the rights.

♢ **Economic Commitment** is a promise or obligation of an economic *Agent* to perform an economic *Event* in the future. For example, line items on a sales order represent commitments to sell goods. Lack of resources leads to unmet demands and, while reflecting the SLA violations, leads to financial consequences and penalties.

♢ **Economic Contract** is a collection of increment and decrement commitments and terms. Thus, the contract can specify what should happen if the commitments are not fulfilled.

In REA, business processes are the orchestration of events that can be triggered by agents affecting the resources. Resources are exchanged through these processes. The notion of stockflow is used to specify in what way an economic event affects a resource. REA identifies five stockflows: *Produce*, *Use*, *Consume*, *Give* and *Take*. For instance, the *Deployment* process of the *Vendor* specifies an *outflow* of *Resources* and *inflow* of *Cash* to the *Vendor*. The model of the *Usage* process from the perspective of the client agent is a mirror image of the vendor's *Deployment* process. The *Usage* pattern of the client specifies the *inflow of Resource* and *outflow of Cash* from the client.

4 TOSCA – The Output Model

The Topology Orchestration Specification for cloud Applications (TOSCA) language introduces a grammar for describing service templates by means of Topology Templates and Plans. The root of a TOSCA service is the Service Template. The Service Template contains a directed graph that represents the structure of

the service called a Service Topology. Every *service template* has at least one *service topology*. The topology graph is composed of nodes and edges. Edges in a directed graph are links with a direction from node to node. The edges in a Service Topology graph are binary relationships between nodes. The nodes represent the logical components of the service. These nodes and relationships are templates that are patterns for the real nodes and relationships instantiated in a deployed service. Plans orchestrate various aspects of a service life cycle. The TOSCA specification defines *Build* plans and *Termination* plans. Build Plans orchestrate the deployment and installation of a service. Termination Plans orchestrate decommissioning of a service. Designers of TOSCA-based applications can add plan types as needed. The designers can benefit by work-flow notations such as BPMN or BPEL. In our CREAM model, TOSCA embodies the cloud composite application design and its elasticity specifications directly derived from the business requirements model using REA.

5 Mapping REA to TOSCA

In this section we describe the mapping from a REA model to TOSCA artifacts. Before we delve into the details of modeling and implementation, it is reasonable to focus on the underlying approaches as we have taken on the mapping process to provide a holistic view about the source model (REA) and target (TOSCA) artifacts. Our approach is twofold: first, we proceed with the conceptual mapping from a meta-level perspective. Second, we define the mapping rules of the two models supported by their implementation scripts in the tool.

5.1 Conceptual Mapping

A mapping from the REA business modeling language to the TOSCA artifacts is a first step in the progress of developing business-oriented cloud applications. This section formulates such a mapping. To define a mapping, we first discover the most suitable matches for REA concepts in TOSCA, then we formulate this connection in rules which will be formalized further in the tooling. We start with the eight concepts derived from the REA as core concepts. As listed in Table 1, we identified the following eight rules.

5.2 Mapping Rules (M.R.)

$\diamondsuit$ **M.R.1**: ***Resource***, indicate things that are affected or exchanged in processes. For cloud applications, software services or infrastructure resources express the same semantics. It can be specified by *nodeTemplate* and *nodeType* elements in TOSCA. For instance, a *nodeType* of *ApacheWebServer* can be instantiated by a *nodeTemplate* of *MoodleAppServer*.

$\diamondsuit$ **M.R.2**: ***Event***, is nested within an economic *Exchange*. These events are initiated by *Agents* affecting a *Resource*. In TOSCA, the *nodeTypes* has element

Table 1. Mapping Rules from REA model to TOSCA artifacts

No	Rules	REA Concepts	TOSCA Concepts
1	Resource	Economic Resource	Node Tempalate
2	Event	Economic Event	Interface Operation
3	Exchange	Economic Exchange	Relations / Plans
4	Entity	Economic Agent	Roles
5	Contract	Contract / Commitment	Policy Types
6	Duality	Exchange Duality	Relation Types
7	Links	Stockflow, Inflow, Outflow	Relations Types
8	Pack	Typification, Grouping	Service Templates

of *Interfaces* in which each interface includes some *Operations*. For instance, releasing or allocating storage resource unit from/to a VM.

♢ **M.R.3**: ***Exchange***, is a value or resource *Exchange* with pair of economic *Events* linked by *Duality* relationship. It is mapped to TOSCA *relationType* and *plans* which defines the process models that are used to manage the application life-cycle. In TOSCA, a plan is a set of operations exposed in a sequence flow by the service template. Both concepts contain the business transactions, resource exchange, events, and agents that are necessary to fulfill the business goal. The typical TOSCA plans are *buildPlans*, *terminationPlans* and can be extended to *modificationPlans*.

♢ **M.R.4**: ***Entity***, is basically an economic unit or an *Agent* representing an actor and therefore mapped to *Role* in TOSCA plans. The mapping is logical since both concepts share the same semantics. TOSCA roles are oriented on three actors of cloud service *Developer*, *Provider* and *Consumer*. An economic agent in REA and a role in TOSCA are both actors with an interest in a collaboration. TOSCA *typeArtifact*, *artifactDeveloper* and *applicationArchitect* are the specialization of the service developer role. Cloud *service provider* hosts and operates the application to be used by the *service consumer*.

♢ **M.R.5**: ***Contract***, details an agreement reflected in an economic *Event*. The resource delivery is governed by an associated *Contract*, composed of set of *Commitments*. An economic contract comprises agreements, rights and terms made among agents. Commitment fulfills the *exchange-reciprocity* application. In TOSCA, the commitments can be declared by the use of *Policy Types* and *AppliesTo* element. A policy type can express the resource intended behavior or the Quality of Service (QoS) that a *nodeType* is about to expose. A TOSCA Policy can also express diverse things like monitoring behavior, payment conditions, scalability, or availability, for instance. Policies can inherit and apply properties by *derivedFrom* and *appliesTo* elements. Thus a relevant policy type can show the specified behavior of a resource in a *Contract*.

♢ **M.R.6**: ***Duality***, also nested within an economic *Exchange* and the *Event* holding this association triggers the resource exchange. Duality can be used to model many-to-many relationships between any two resources. This allows *Give & Take* operations to increase or decrease the amount of resource allocation. Duality implements the elasticity behavior of the cloud application. Thus, the messaging among the resources should be paired via a duality relationship to bind events together with the resource exchange. For instance, *Request & Response*, *Demand & Allocate*, *Service Acquisition & Service Provision* and *Pay_per_resource_usage* can be considered as cloud use-cases of *Duality* concepts. In this sense, *Duality* is mapped to TOSCA *relationType* that identifies the corresponding relation of a service provisioning event to a specific request and payment subsequently.

♢ **M.R.7**: ***Link***, denotes the semantics behind the links among service encompassed components. The *Stockflow* association denotes the flow of resource exchange triggered by an economic events like increment or decrement resource allocation. The relationship between an increment event and a resource is called *inflow* and the relationship between a decrement and a resource is called *outflow*. For instance, in vendor's *sales* process, the exchange will represent an *outflow* of resource and an *inflow* of cash in return. In TOSCA, the relationship specifies the semantics between nodes of *sourceElement* and *targetElement* in a topology template. The REA relations can be mapped to the TOSCA *relationTypes* like *dependsOn*, *hostedOn* and *deployedOn* concerning the context.

♢ **M.R.8**: ***Pack***, is a course or principle of composition action, adopted by *Grouping* and *Typification* abstractions in the REA application model. Typification implements *a-kind-of* element, grouping realizes *a-member-of* applications. This forms a composite application which will be deployed under certain policies. Hybrid association of *Types* and *Groupings* defines the *Policy Layer* on top of the *Operation Layer* in the model. In TOSCA, a policy type defines the constraints of a property, i.e. data types, allowed values, obligations and authorization requirements in a corresponding template.

6 Implementation: CREAM Tool Support

The aim of this toolkit is to provide a framework to facilitate the modeling and deployment of cloud based applications. Our toolkit provides a web interface which hides and abstracts away the cloud implementation details to business developers. CREAM captures the system requirements and their relationships, then builds the cloud application topology in TOSCA. The CREAM is a Java-based web application which is developed in WSO2 Developer Studio[7]. We used *Maven* to resolve its dependencies and deployed CREAM on WSO2 Application Server. Cloud resources are stored in WSO2 Governance Registry in compliance with TOSCA standard. All resources and artifacts are located in "/*cream*" path

[7] `http://wso2.com/products/developer-studio`

in the registry and categorized in two collections: (i) TOSCA Templates: this collection contains cloud and REA resources. For instance, *Instructor* is mapped to a TOSCA *NodeType* which is located in human resources category (HuaaS). For each resource and collection in "*/cream/ToscaTemplates*", a title is set in registry that will be displayed in CREAM Tool canvas, otherwise the name of the resource will be used. (ii) CSAR: the cloud topologies designed by business and application developers will be stored in this collection. Each designed topology is a TOSCA XML file named with a UUID and contains a *ServiceTemplate*. This contains all required information about services and resources requested by the user.

6.1 CREAM Architecture

Now, we detail the architecture. We developed the CREAM Toolkit based on a Model-View-Controller (MVC) design pattern. MVC framework is designed around a *DispatcherServlet* that dispatches requests to handlers. In CREAM, Dispatcher servlet is responsible to handle requests and responses. It delegates requests to controller (i.e., class *CloudApplicationDesignerController*). Controller class is identified by *@Controller* annotation and has methods to handle incoming requests. Each URL is mapped to a method annotated with *@RequestMapping*. This method executes the user requests, generates a model object and returns it to dispatcher. Dispatcher send models to view template which is responsible to render response. Finally dispatcher returns rendered response to user. For the sake of brevity, we only describe the packages and classes to clarify the CREAM architecture as illustrated in Fig 2.

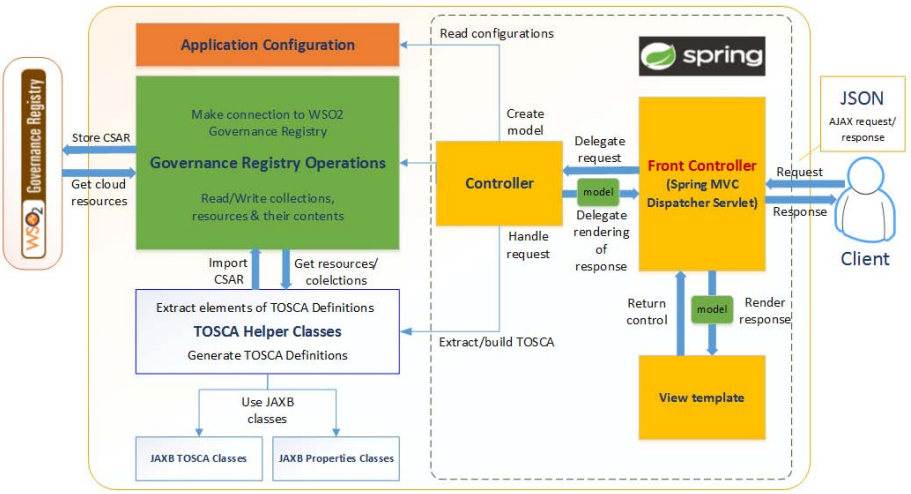

Fig. 2. Cloud REA Model (CREAM) architecture

6.2 Package Description

In this section, we describe the packages, their bundled classes, and implemented interfaces to support the CREAM architecture.

♢ **Package org.cream.commons**
This package includes exception classes, simple classes for Jakson *ObjectMapper* and other helper classes which are common in whole application. Its core classes are *ApplicationConfiguration*, *ServletContextHelper*, *ResourceObjectMap*, and *DesignedApplicationObjectMapItem*. The *ApplicationConfiguration* is responsible to read configuration file and make its entries accessible by other components of the application. The *ApplicationConfiguration* uses the *ServletContextHelper* class to find the real path of the configuration file. Both classes are designed using Singleton pattern.

♢ **Package org.cream.tosca.model**
This package contains JAXB generated classes from TOSCA XML schema (XSD). It also contains a sub-package `org.cream.tosca.model.properties` which includes JAXB generated classes for our defined properties schema. There are several sub-packages such as `org.cream.tosca.model.properties.amazonec2` whereas each package contains JAXB generated classes from a specific properties XML schema file. We use *Properties* element in TOSCA *NodeTemplate* to store specifications of each resource. We have defined these properties elements for each resource with XML schema. For each XML schema, we have generated corresponding classes using Java API JAXB. All packages in `org.cream.tosca.model.properties` corresponds to one schema.

♢ **Package org.cream.wso2.greg**
This package contains helper classes to connect to WSO2 Governance Registry and to retrieve resources and collections. Class *GovernanceRegistryConnector* is responsible to make connection to WSO2 Governance Registry. Method *getRemoteRegistry* returns an instance of class *RemoteRegistry* since the registry data retrieval APIs are defined here. Class *GovernanceRegistryReader* is responsible to read and write resources.

♢ **Package org.cream.tosca.loader**
Classes of this package works with JAXB generated classes. They extract TOSCA elements from TOSCA files and generate TOSCA Definitions and CSAR files. Class *JAXBMetaDataExtractor* uses Java *Reflection* API to extract properties' element names from JAXB property classes. Class *ToscaFileReader* marshals TOSCA Definitions from the given *InputStream*. It also provides a few helper classes for entire application to retrieve needed information about a TOSCA XML file. Class *ToscaBuilder* is responsible to generate final TOSCA definition object from user-defined topology. Finally this class converts the generated TOSCA Definitions to its XML string and stores it in WSO2 Governance Registry.

7 Conclusion and Outlook

So far, we have used the REA model to specify the business requirements, constraints and rules for building cloud applications. In support of our approach, we developed the CREAM tool in which, initially does the conceptual mapping and build the TOSCA-based cloud application. As an outlook, our future work includes further extension to the CREAM tool that can also support the REA's structural and behavioral business patterns[15] at policy, operational and aspect layers to provide a more holistic coverage of the various perspectives relevant to application development process. Summarizing, we envision cloud REA Model as a potential cloud value modeling framework for building business-driven cloud applications.

References

1. Papazoglou, M.P.: Cloud blueprints for integrating and managing cloud federations. In: Heisel, M. (ed.) Software Service and Application Engineering. LNCS, vol. 7365, pp. 102–119. Springer, Heidelberg (2012)
2. Osterwalder, A., Pigneur, Y., Tucci, C.L.: Clarifying business models: Origins, present, and future of the concept. Communications of the Association for Information Systems 16, article 1 (2005)
3. Gordijn, J., Akkermans, H.: e3-value: Designing and evaluating ebusiness models. IEEE Intelligent Systems 16(4), 11–17 (2001)
4. Mccarthy, W.E.: The rea accounting model: A generalized framework for accounting systems in a shared data environment. The Accounting Review 57(3), 554–578 (1982)
5. Iso: Information technology - business operational view - part 4: Business transaction scenarios, iso/iec 2007, iso 15944-4 (2007)
6. Oasis, un/cefact: ebxml - technical architecture specification, version 1.4 (February 2001)
7. Mohammed, A.B., Altmann, J., Hwang, J.: Cloud computing value chains: Understanding businesses and value creation in the cloud. In: Economic Models and Algorithms for Distributed Systems, Autonomic Systems, pp. 187–208. Birkhäuser, Basel (2010)
8. Gordijn, J., Yu, E., van der Raadt, B.: E-service design using i* and e3value modeling. IEEE Software 23(3), 26–33 (2006)
9. Ramel, S., Grandry, E., Dubois, E.: Towards a design method supporting the alignment between business and software services. In: 33rd Annual IEEE International Computer Software and Applications Conference, COMPSAC 2009, vol. 1, pp. 349–354 (2009)
10. Gailly, F., Geerts, G.: Frederik Gailly and Guido Geerts. Formal definition of business rules using rea business modeling language. In: Proceedings of the 7th International Workshop on Value Modeling and Business Ontology, p. 7 (2013)
11. Schuster, R., Motal, T., Huemer, C., Werthner, H.: From economic drivers to B2B process models: A mapping from REA to UMM. In: Abramowicz, W., Tolksdorf, R. (eds.) BIS 2010. LNBIP, vol. 47, pp. 119–131. Springer, Heidelberg (2010)
12. Sonnenberg, C., Huemer, C., Hofreiter, B., Mayrhofer, D., Braccini, A.: The REA-DSL: A domain specific modeling language for business models. In: Mouratidis, H., Rolland, C. (eds.) CAiSE 2011. LNCS, vol. 6741, pp. 252–266. Springer, Heidelberg (2011)

13. Roelens, B., Lemey, E., Poels, G.: A service science perspective on business modeling. In: Proceedings of the 6th International Workshop on Value Modeling and Business Ontology, p. 8 (2012)
14. Poels, G.: The resource-service-system model for service science. In: Trujillo, J., et al. (eds.) ER 2010. LNCS, vol. 6413, pp. 117–126. Springer, Heidelberg (2010)
15. Hruby, P.: Model-Driven Design Using Business Patterns. Springer-Verlag New York, Inc., Secaucus (2006)